Designing Digital Services  
with Cryptographic Guarantees  
for Data Security and Privacy  

Von der Fakultät für Mathematik, Informatik und Naturwissenschaften  
der RWTH Aachen University zur Erlangung des akademischen Grades  
eines Doktors der Naturwissenschaften genehmigte Dissertation

vorgelegt von

Diplom-Informatiker  
Jan Henrik Ziegeldorf  
aus Schwerte, Deutschland

Berichter:  
Prof. Dr.-Ing. Klaus Wehrle  
Prof. Dr. rer. nat. Björn Scheuermann

Tag der mündlichen Prüfung: 08.12.2017

Diese Dissertation ist auf den Internetseiten der Universitätsbibliothek online verfügbar.
Reports on Communications and Distributed Systems

edited by
Prof. Dr.-Ing. Klaus Wehrle
Communication and Distributed Systems,
RWTH Aachen University

Volume 16

Jan Henrik Ziegeldorf

Designing Digital Services with Cryptographic Guarantees for Data Security and Privacy

Shaker Verlag
Aachen 2018
Abstract

In the past two decades, tremendously successful digital services have been built that collect, process, and monetize massive amounts of personal user data, up to the point where data is proclaimed the oil of the 21st century. Along come serious threats to data security and privacy that significantly increase the demand for effective protection, e.g., as manifested in the growth of encrypted Internet traffic. Communication security protocols, however, protect data against external attackers and do not address the root cause of almost all privacy threats, the need to share sensitive data with third parties. These third parties may illicitly process data beyond its original purpose of collection or be hacked and forced to provide data access. Countering these threats requires the development of Privacy Enhancing Technologies that complement or replace traditional communication security protocols.

We identify Secure Multiparty Computation (SMC) as a rigorous approach not only to provide data security and privacy protection, but even to reconcile privacy interests with seemingly adverse public and business interests. However, the potential of SMC is foremost on the theoretical level – it is often dismissed for being too inefficient and impedimentary for real-world applications. This thesis bridges the gap between the theoretical strength of SMC and the feeble realization of its potential in practice. To this end, we conduct a qualitative and quantitative analysis of SMC frameworks and abstract three research challenges: i) Extending the functionality and ii) increasing the efficiency of SMC as well as iii) customizing it to challenged environments. We choose a use case-driven research methodology to address these questions, which allows us to motivate and validate all our contributions in practice.

First, we motivate the problem of financial privacy in cryptocurrencies and propose decentralized mixing as a solution. We recognize the advantages of securing mixing operations with SMC and contribute secure protocols to technically realize our novel approach. As a result, our mixing system achieves stronger security and privacy guarantees than prior works while remaining highly scalable and fully compatible with the prevalent designs of decentralized cryptocurrencies such as Bitcoin.

Second, we propose efficient SMC designs for different classification algorithms to address data security and privacy issues in pattern recognition and machine learning. The evaluation of our classifiers shows that they are secure, accurate, and outperform the state of the art. We demonstrate three real-world use cases that prove applicability of our classifiers but also motivate their deployment in challenged environments. Thus, we present two additional approaches, bandwidth optimizations and secure outsourcing, to bring our secure classifiers to these scenarios.

Finally, we investigate secure outsourcing as a general strategy to customize SMC to challenged deployment and operation scenarios by the example of computing set intersections, a universal building block in many real-world applications and a well studied SMC problem. We present efficient schemes with negligible overheads for the outsourcers and demonstrate their applicability in two comprehensive case studies, privacy-preserving crowd-sensing and genetic disease testing in the cloud.

In summary, the contributions made in this thesis widen the technical solution space for practical data security and privacy protection in data-driven digital services.
Kurzfassung


Zuerst zeigen wir Privatsphäreprobleme im Bereich digitaler Währungen auf und untersuchen dezentralisierte Mix-Systeme als Lösung. Wir erkennen und motivieren die Vorteile, solche Systeme durch SMC abzusichern, und entwerfen anschließend die erforderlichen Protokolle. Unser Ansatz gewährleistet nicht nur stärkere Sicherheits- und Privatsphäregarantien als bisherige, sondern bleibt dabei skalierbar sowie vollständig kompatibel zu den Designs vorherrschender Kryptowährungen wie z.B. Bitcoin.


Insgesamt erweitern unsere Beiträge den technischen Lösungsraum praktikabler Daten- und Privatsphäreschutzmechanismen für datengetriebene digitale Services.
Acknowledgements

So many people have directly or indirectly influenced my dissertation in so many different ways that I cannot name them all. I am deeply grateful to all colleagues, friends, and family, who accompanied me along this journey. From its beginnings as a diploma student at COMSYS, over to Philips Research, Eindhoven, and back to COMSYS for good, as well as through all the little "detours" that made work and life as a PhD student so much more interesting and worthwhile – I have enjoyed your company, wisdom, and support both professionally and privately and I sincerely hope you stick around for what is to come.
Contents

1 Introduction ................................ 1
   1.1 Analysis of Problems and Challenges ................. 4
   1.2 Research Questions and Methodology .................. 8
   1.3 Contributions of this Thesis ........................ 9
       1.3.1 Attribution of our Contributions .................. 12
   1.4 Outline .................................... 13

2 Secure Multiparty Computation .......................... 15
   2.1 From Distributed Computing to SMC .................... 15
   2.2 Model and Security Definitions ........................ 17
       2.2.1 Attacker Model .............................. 18
       2.2.2 Limitations of Security Definitions .............. 20
   2.3 Generic Secure Protocols ............................ 21
       2.3.1 Oblivious Transfer ............................ 22
       2.3.2 Secure Computation of Boolean Circuits .......... 24
       2.3.3 Secure Computation of Arithmetic Circuits ....... 29
       2.3.4 Notation ...................................... 34
   2.4 Instantiations and Applications ....................... 36
       2.4.1 Frameworks and Libraries ....................... 36
       2.4.2 Languages and Compilers ........................ 37
       2.4.3 Benchmarks and Applications ..................... 38
   2.5 Summary ....................................... 39

3 Decentralized Mixing of Digital Currencies ............. 41
   3.1 Motivation ...................................... 41
   3.2 Problem Analysis .................................. 43
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.4</td>
<td>Secure Classification Framework and Designs</td>
<td>105</td>
</tr>
<tr>
<td>4.4.1</td>
<td>Representation of Real Numbers</td>
<td>106</td>
</tr>
<tr>
<td>4.4.2</td>
<td>Hyperplane Classifier</td>
<td>109</td>
</tr>
<tr>
<td>4.4.3</td>
<td>Artificial Neural Networks</td>
<td>111</td>
</tr>
<tr>
<td>4.4.4</td>
<td>Naive Bayes</td>
<td>114</td>
</tr>
<tr>
<td>4.4.5</td>
<td>HMM Forward</td>
<td>117</td>
</tr>
<tr>
<td>4.4.6</td>
<td>HMM Viterbi</td>
<td>121</td>
</tr>
<tr>
<td>4.4.7</td>
<td>Security Discussion</td>
<td>123</td>
</tr>
<tr>
<td>4.4.8</td>
<td>Evaluation</td>
<td>125</td>
</tr>
<tr>
<td>4.4.9</td>
<td>Use cases</td>
<td>136</td>
</tr>
<tr>
<td>4.4.10</td>
<td>Summary and Future Work</td>
<td>146</td>
</tr>
<tr>
<td>4.5</td>
<td>Secure Classification and Pattern Recognition in Constrained Environments</td>
<td>148</td>
</tr>
<tr>
<td>4.5.1</td>
<td>Towards Bandwidth-optimized Secure Computations</td>
<td>148</td>
</tr>
<tr>
<td>4.5.2</td>
<td>Secure Outsourcing to Untrusted Computation Clouds</td>
<td>159</td>
</tr>
<tr>
<td>4.6</td>
<td>Conclusion</td>
<td>165</td>
</tr>
<tr>
<td>5</td>
<td>Outsourced Private Set Intersection</td>
<td>167</td>
</tr>
<tr>
<td>5.1</td>
<td>Motivation</td>
<td>168</td>
</tr>
<tr>
<td>5.2</td>
<td>Problem Analysis</td>
<td>170</td>
</tr>
<tr>
<td>5.2.1</td>
<td>Problem Statement</td>
<td>170</td>
</tr>
<tr>
<td>5.2.2</td>
<td>Related Work</td>
<td>173</td>
</tr>
<tr>
<td>5.2.3</td>
<td>Our Contributions</td>
<td>178</td>
</tr>
<tr>
<td>5.3</td>
<td>Outsourced Private Set Intersection</td>
<td>179</td>
</tr>
<tr>
<td>5.3.1</td>
<td>Outsourcing to Many Peers</td>
<td>179</td>
</tr>
<tr>
<td>5.3.2</td>
<td>Outsourcing to Two Peers</td>
<td>181</td>
</tr>
<tr>
<td>5.3.3</td>
<td>Outsourcing to a Single Peer</td>
<td>183</td>
</tr>
<tr>
<td>5.3.4</td>
<td>Evaluation and Discussion</td>
<td>187</td>
</tr>
<tr>
<td>5.3.5</td>
<td>Summary and Future Work</td>
<td>193</td>
</tr>
<tr>
<td>5.4</td>
<td>Case Study: Crowd-Sensing</td>
<td>194</td>
</tr>
<tr>
<td>5.4.1</td>
<td>Motivation</td>
<td>195</td>
</tr>
<tr>
<td>5.4.2</td>
<td>Problem Statement</td>
<td>196</td>
</tr>
<tr>
<td>5.4.3</td>
<td>TraceMixer Design</td>
<td>199</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
<td>------</td>
</tr>
<tr>
<td>5.4.4</td>
<td>Evaluation and Discussion</td>
<td>203</td>
</tr>
<tr>
<td>5.4.5</td>
<td>Summary</td>
<td>208</td>
</tr>
<tr>
<td>5.5</td>
<td>Case Study: Genetic Testing</td>
<td>209</td>
</tr>
<tr>
<td>5.5.1</td>
<td>Motivation</td>
<td>210</td>
</tr>
<tr>
<td>5.5.2</td>
<td>Problem Statement</td>
<td>211</td>
</tr>
<tr>
<td>5.5.3</td>
<td>Secure Outsourced Queries over Genomic Data</td>
<td>212</td>
</tr>
<tr>
<td>5.5.4</td>
<td>Evaluation and Discussion</td>
<td>214</td>
</tr>
<tr>
<td>5.5.5</td>
<td>Summary and Future Work</td>
<td>222</td>
</tr>
<tr>
<td>5.6</td>
<td>Conclusion</td>
<td>223</td>
</tr>
</tbody>
</table>

6 Conclusion 225

6.1 Contributions 226
6.1.1 Secure Decentralized Mixing of Digital Currencies 226
6.1.2 Privacy-preserving Pattern Recognition and Classification 227
6.1.3 Outsourced Private Set Intersection 227

6.2 Future Work 228
6.3 Concluding Remarks 230

A Additional Evaluation of Selected Building Blocks in SHIELD 231

Glossary 239

Bibliography 239
Introduction

Privacy is recognized as a fundamental human right in the 1948 Universal Declaration of Human Rights [udh48] and is anchored in the constitutional law of most modern states. Historically, privacy notions evolved from territorial and bodily privacy over media and communication privacy, best characterized by Brandeis as “the right to be let alone” [WB90], to the now predominant notion of information privacy [Lan01], defined in 1968 by Westin as “the right to select what personal information about me is known to what people” [Wes68]. Although originally referring non-digital environments, Westin’s definition is still popular and adequate today as digitalization taps more and more sources of personal data about people’s everyday lives. Indeed, we witness a growing reality of service providers that build extremely lucrative businesses upon accumulating, mining, and selling their users’ data – often without their knowledge and contra their expectations [IL17]. This large scale collection of personal data causes privacy risks which have, inevitably, manifested in numerous high profile incidents [Thi16, Men16, Fra16]. In reaction, users have become concerned about where their personal data is stored, how it is processed by third party services, as well as whom their personal data is re-distributed to [Rai16].

The rising demand for digital privacy can be observed most clearly by the adoption of Transport Layer Security (TLS), e.g., the volume of TLS protected Internet traffic has increased from 5% to 35% within only two years (July 2012 to 2014) [NFL+14]. TLS [DR08] and similar approaches [KS05, YL06] protect the confidentiality, integrity, and authentication of communications against external attackers. In contrast, existing and emerging privacy threats are due to internal attackers, e.g., service providers that have been hacked, seized, or compromised otherwise [Lad14, Men16, Fra16]. These threats are thus incongruent to or go far beyond the classical protection goals in communication security and have emancipated Privacy Enhancing Technologies (PETs) as a fully independent research field.

The main cause for privacy threats in today’s digital landscape is that users are frequently required to share sensitive data with unknown third parties – once data is shared, users need to trust them not to misuse their data secure and keep it secure.
Most proposed PETs set out to address this fundamental issue in one way or the other, e.g., through privacy policies and access control [BBL05, KCLC07], anonymization [RGFN08], or data aggregation and perturbation [Swe02, BA05]. Shortcomings have, however, been documented for these approaches [Swe02, NS08, EEJAM11], among which we identify two root causes: First, the fundamental problem of sharing sensitive data with third parties is not eradicated but only shifted, e.g., to anonymization proxies [Haf06]. Second, ad-hoc assumptions about the capabilities and knowledge of potential attackers are made that later prove too optimistic or wrong as progress in technology or the availability of further background information create unforeseen attack vectors. The authors of [Swe02, NS08, Ohm09], e.g., show how to reidentify individuals in supposedly anonymous public databases using side-knowledge that was overlooked or not yet available during the initial anonymization process. Importantly, these are not simple matters of human negligence but deeply rooted design flaws that require a different approach to the design of PETs.

Interestingly, both issues can be addressed using Secure Multiparty Computation (SMC), a long-studied problem in cryptography for which theoretic feasibility results were presented already in the late 1980’s independently by Yao [Yao86] and Goldreich et al. [GMW87]. Informally stated, SMC considers how two or more parties can collaboratively compute a joint function of their individual private inputs without revealing to each other or any third party anything but the final result – even if parties collude or possess additional side knowledge. Since the joint functionality is not limited in any ways other than being computable, SMC is conceptually a promising approach to rigorously protect privacy in digital services. For example, a personalized web service (e.g., for genetic disease testing [ZSG14, DCFGT12, 23a]) could be realized as an SMC between a user who inputs sensitive personal data (e.g., her genome) and obtains the result from the service provider who remains oblivious of the user’s input and the computed result (e.g., a diagnosed illness).

While the early theoretic results for SMC are conceptionally powerful, SMC was long believed to be too inefficient in practice and it took almost two decades of progress in computer performance for a first real-world application to emerge. In January 2008, SMC was used to determine among the confidential bids of 1200 danish farmers the market clearing price for 25,000 tons worth of sugar beet production contracts [BCD+09]. The corresponding SMC protocol (consisting of simple arithmetic operations and comparisons) ran for 30 minutes – without privacy, the same task could be computed in seconds on the unencrypted bids. This example demonstrates two core insights about SMC: First, SMC entails very high overheads since all computations must be carried out under encryption. Second, despite these high overheads, certain use cases are improved or only enabled by rigorous protection mechanisms such as SMC. In the danish sugar beet auctions, SMC not only obviated the need to pay a notary and trust him to act as a neutral unattached auctioneer, SMC even ensured that no one learned the confidential bids, not even a paid trusted notary.

To motivate the importance of SMC beyond this example, we have conducted a qualitative analysis of existing and emerging security and privacy threats [ZGMW14]. Our study focuses on the Internet of Things but similar results that validate and complement ours have been presented for related domains, e.g., in participatory sensing [Chr16] and cloud computing [CZ12, FLR13]. From these results, we distill major technological and societal trends that cause and aggravate security and privacy risks and briefly point out if and how SMC helps to address them.
T1 - Proliferation of mobile devices and cloud services: The increasing adoption of mobile devices and applications leads to the collection of more and more sensitive personal data, e.g., mobility, activity, and health data [LML’10]. Collected data is typically transferred to distant cloud servers for convenient storage, processing, and visualization [FLR’13] which, unfortunately, requires users to give up data sovereignty and personal privacy [CRKH’11, CZ’12]. To counter associated risks, SMC enables encrypted cloud services that store and process users’ data securely in an encrypted domain [BNSS’11, LATV’12, ZPH’17].

T2 - Data-driven research and businesses: Over the last two decades, innovative research and tremendously successful businesses have been based on and fueled by personal data [CML’14]. This trend continues and drives the digitalization even of critical sectors that mandate strongest data protection such as healthcare and genomics [CV’15, SLF’15]. In this context, SMC not only offers the technical means to reconcile research and business interests with security and privacy requirements [JZW’14, TJW’16], but even promises to enable the digitalization of new sectors that are thus far obstructed by legislation, regulation, and compliance requirements [CK’08, BTW’12, ABPP’16].

T3 - Data leaks and forced access: The vast amount of sensitive data processed and stored in central cloud servers and databases is at an increasing risk: Massive breaches and data leaks due to hackers, insiders, or human errors are the daily fare [Pri’17, Men’16, Thi’16], aggravated by governments that – in the name of public safety – demand practically unconstrained access to confidential user data [Lad’14, AAB’15]. Such compromised end systems are, however, a threat that is not considered by classical communication security protocols [RK’03]. In contrast, SMC allows service providers to remain oblivious of users’ critical data thus protecting both themselves and their users against breaches and forced access from in- and outsiders [SS’13, ZPH’17].

T4 - Opaque legislation: The role and impact of data protection legislation is almost schizophrenic. On the one hand, legislation is frequently changed, bent, and broken, not least by governments and private sector intelligence services, e.g., as made public by the Snowden leaks [Lyo’14]. On the other hand, obliging companies and public offices struggle to comply with applicable laws and regulation [va’14]. Both issues are two sides of the same coin: the lacking means of enforcement that SMC can provide through technical mechanisms.

In summary, we face huge digital security and privacy challenges that are only going to grow given the discussed technological and societal developments. Our observations clearly motivate the need, effectiveness, and benefits of SMC to face these challenges, yet SMC remains to date a niche technology that is often prematurely dismissed for being inefficient, inapplicable, and impractical [ABPP’16]. This thesis is motivated by the gaping mismatch between the theoretical strength and observed potential of SMC, on the one hand, and the actual application and utilization of SMC for security and privacy protection in real-world digital services, on the other. On the highest level, the goal of this thesis is thus to significantly contribute to the ongoing efforts of establishing SMC as a practical security and privacy protection framework. This is driven by the vision of SMC that does not prevent but enable future digital services by reconciling public, private, and business interests.
1. Introduction

1.1 Analysis of Problems and Challenges

In this section, we define the problem space and scope of this thesis. To this end, we analyze the open research area of designing digital services with rigorous cryptographic security and privacy protection based on SMC. Our goal is to formulate concrete open research questions and challenges for SMC that need to be addressed on the path out of its niche existence towards technical maturity and adoption.

To a major part, the results presented in this section are based on and motivated by our independent qualitative and quantitative study of secure computation frameworks [ZMHW15] as well as an independent analysis of the maturity of SMC presented by Archer et al. [ABPP16]. These works reveal practical shortcomings of existing SMC frameworks that explain the hesitant adoption of SMC from a technical point of view. To keep the following discussion both general and concise, we abstract from the specific frameworks and applications presented in theses works with the goal to regard SMC holistically.

We extract from [ZMHW15, ABPP16] a hierarchy of needs that remain (partly) unfulfilled and hinder the technical maturity and adoption of SMC as shown in Figure 1.1: At the base, we have observed four major technological and societal trends (T1–T4) that create and aggravate problems and concerns for digital security and privacy to which SMC presents a promising solution. Based on these trends, we identify concrete use cases that first motivate and demonstrate the benefits of SMC and later serve as validation of our designs and obtained results. Realizing these use cases challenges and exceeds the current state of the art and requires to extend functionality (Q1) and increase efficiency (Q2) of SMC to push the boundaries of technical feasibility. In addition to technical feasibility, the constraints of real-world deployments and operations (especially those involving mobile devices) hamper the proliferation of SMC protocols and technology and require their customization,
1.1. Analysis of Problems and Challenges

to different cost metrics, network dynamics, or constrained resources of devices and networks (Q3). Finally, SMC technology must be standardized in order to be embedded in future digital services and data protection legislation. We emphasize that the latter stage of stabilization can only be approached when SMC has reached a reasonable measure of technical maturity and practical adoption which is beyond the contentual and temporal scope of this thesis.

In the following, we briefly analyze each of the three stages within the scope of this thesis, i) motivation and validation, ii) technical feasibility, and iii) proliferation, and distill pressing research questions.

Stage 1: Motivation and Validation

SMC offers a rigorous and generic protection framework for computations over distributed sensitive data. However, security and privacy protection is never an end in itself. Any security and privacy technology requires an application context and is valued by the contributed benefits to it. We have previously discussed four trends that create promising application areas for SMC. These are long-term perspectives that, arguably, must be approached gradually, through a series of much smaller nonetheless compelling use cases.

Finding such use cases is surprisingly difficult. The majority of SMC research is motivated from the idealistic point of view that privacy is a basic human right and vital ingredient of democratic societies which deserves protection. An important use case of SMC is then to protect user’s privacy in computations on sensitive data that has the potential to harm them if disclosed. Unfortunately, the motivation of most service providers is less idealistic and privacy protection is not necessarily in their economic interest. Instead, applicable data protection legislation is perceived as an impediment to productivity and profit and treated with minimal attention and effort [Ran17] – even in business-to-business scenarios such as targeted advertising, where simple yet potentially ineffective techniques have been favored over rigorous but more expensive approaches such as SMC [PSSZ15]. This negligence is aggravated by the fact that the concept of SMC as well as its guarantees and limitations are difficult to grasp for non-experts. In consequence, the potential of SMC is rarely recognized by practitioners in potential application areas.

While not a technical research challenge in itself, identifying compelling use cases that are technically feasible and non-zero-sum is an important first step as they drive the development and adoption of SMC and later serve to validate the applicability of obtained results. Although hard to find, the examples of the danish sugar beet auctions [BCD+09] and a few other success stories, e.g., in advertising [Kre17] or financial data analysis [BTW12], show that diverse use cases exist. The General Data Protection Regulation (EU 2016/679) coming into effect in the European Union by 2018 couples the height of fines to a company’s volume of sales which will provide strong additional incentives for the adoption of SMC.

\[\text{We borrow and slightly abuse this notion from the field of game theory to describe scenarios in which the adoption of SMC is beneficial for all involved parties.}\]
1. Introduction

Stage 2: Technical Feasibility

A core insight of [ZMHW15, ABPP16] is that the technical realization of many envisioned use cases will require extended functionality and increased efficiency of SMC. In our study [ZMHW15], we observed that the five studied SMC frameworks fail in a large fraction of the defined evaluation benchmarks either due to functional limitations or prohibitive runtimes and memory consumption. We discuss the two challenges of extending functionality and increasing efficiency in more detail.

Extending functionality

As we observe in [ZMHW15], only a very limited amount of functionality has actually been implemented in the surveyed SMC frameworks, i.e., mostly low level logical operations and basic arithmetic over the integers, and implementing new functionality in a secure and efficient manner is challenging. Although the analyzed frameworks have been extended since our study, e.g., with floating point arithmetic [DDK+15], the overall available functionality is still severely limited compared to what is offered by modern programming languages [ABPP16]. The existing functional limitations currently restrict possible applications of SMC and must be overcome.

There are basically two opposing philosophies for the design of secure protocols for new functionality: We can either build on top of generic SMC protocols or develop special purpose protocols, e.g., as analyzed for the specific task of computing set intersections in [HEK12, PSSZ15]. Both ways involve certain difficulties and disadvantages. Generic SMC protocols often incur unnecessary high overheads, especially when the desired functionality does not fit well with the underlying functional representation of the chosen generic SMC protocol; a bad example is expressing arithmetic tasks in an SMC protocol that is rooted in Boolean logic as recognized in [HSS+10, DSZ15]. In cases where generic approaches are too inefficient or fail, SMC protocols are typically tailored to the specific problem. While special purpose protocols can be highly efficient where generic ones are not, they rarely generalize and must be developed anew for each use case [PSSZ15]. Designing SMC protocols from scratch, however, requires expert knowledge of the low level SMC techniques and building blocks as well as their definitions of security and composability.

In conclusion, developing new SMC protocols for missing functionality presents an important research challenge to improve the technical maturity, applicability, and adoption of SMC. A special challenge is to find sensible trade-offs between generic and special purpose approaches such that newly developed functionality is both efficient and applicable beyond its initial application scope.

Increasing efficiency

Due to ciphertext expansion\(^2\) and oblivious program execution\(^3\), SMC protocols generally involve orders of magnitude higher overheads than executing the equivalent

\(^2\)SMC usually requires some sorts of encryption of inputs and intermediate values which increases message sizes, especially when semantic security requires probabilistic encryption [Pai99, DJN10, KSS13b].

\(^3\)Not knowing the inputs, SMC protocols must execute all possible program branches [LWN+15].
functionality insecurely over plaintexts. We benchmark several SMC frameworks in [ZMHW15] to quantify these effects: The fastest framework, e.g., required almost five minutes for the simple task of multiplying two $15 \times 15$ matrices and many of the more complex benchmarks failed due to prohibitive memory consumption. Since our study, performance of SMC has been greatly improved but still represents a major impediment to the widespread adoption of SMC [ABPP16].

Efficiency of SMCs can be increased in different ways. The most general approach is to improve the generic SMC protocols and backends that are at the base of most SMC applications. However, recent research results indicate that lower bounds may be reached regarding the underlying cryptographic foundations [ZRE15]. An alternative way are algorithmic improvements: While we know the most efficient algorithms on clear texts in many cases, they are not necessarily the best choice for SMC due to the completely different execution model of SMC. For example, conditional branching is very inefficient in SMC such that secure sorting is more efficiently realized using sorting networks in $O(n \log(n)^2)$ than by using one of the well known $O(n \log(n))$ algorithms [HEK12,BA16]. Thus, algorithm design for SMC must be adapted to the strength and weaknesses of the SMC foundation to reach optimal results.

All of the discussed approaches require an in-depth understanding of SMC protocols and algorithm design. We thus conclude with a second research challenge that naturally arises from the previous: improving the efficiency of existing and newly developed SMC functionality. This will be especially important and challenging as foreseen application areas for SMC typically involve large data volumes, e.g., in genetics [JZW+14,TJW+16], machine learning [DGBL+16], or finance [BTW12].

Stage 3: Proliferation

Due to their high processing, communication, and memory overheads, SMC protocols have traditionally been developed and deployed only in fixed setups featuring high powered hosts connected over fast and stable networks. However, emerging applications, e.g., in mobile scenarios [HCE11], the Internet of Things [YZV14], or crowd-computing [MYCH10], challenge traditional SMC protocol designs with dynamic networks, novel interaction patterns, varying cost metrics, and constrained hosts or networks. As we observe in [ZMHW15], most SMC protocols do not adjust well to such challenged environments which hinders the proliferation of SMC.

Different special purpose approaches have been proposed in order to tailor SMC protocols to these new requirements. Prominent approaches include trade-offs, e.g., between processing and communication [DKS+17], or securely outsourcing the execution of SMC protocols to more stable and powered environments [JZW+14,TJW+16]. The latter approach has become attractive with the advent of Fully Homomorphic Encryption (FHE) [Gen09,VDGHV10,BV14] which, however, induces high overheads and requires adaption and fine tuning to the concrete use case [DGBL+16].

Applying SMC beyond its traditional scope clearly challenges existing SMC protocol designs and demands for more flexible approaches. A third research challenge is, thus, the development of SMC protocols that customize to novel, especially challenged deployment scenarios. Since many of these involve mobile users, e.g., crowd-sensing [GYL11] or device-to-device interaction [DSZ14], we especially need to tackle the challenges that mobility and mobile devices present to SMC.
1.2 Research Questions and Methodology

From the challenges discussed in the previous section, we condense three corresponding technical research questions that define the scope of this thesis. We then present our research methodology for answering these questions.

Q1 - Which novel SMC functionality is required and how can it be designed?

Novel application scenarios bring up new functional requirements that demand the extension of existing SMC frameworks and the design of new protocols. Our goal is to contribute new SMC functionality that combines and balances special purpose designs for efficiency with general purpose designs for applicability and thereby to qualitatively and quantitatively extend the suite of available SMC functionality.

Q2 - How can the efficiency of SMC be improved?

To ensure the wide applicability of SMC, especially in data intensive services, the efficiency of SMC must be significantly improved – ideally by orders of magnitude. We aim for algorithmic improvements of existing and the design of novel SMC functionality.

Q3 - How can SMC be customized to constrained environments?

Progressing the use cases, the functionality, and the efficiency of SMC creates new deployment and operational scenarios to which SMC protocols must be customized. We particularly focus on the challenge of tailoring SMC protocols to mobile deployments.

We emphasize that without a clear self-purpose, fate and fortune of SMC technology lies in the actual benefits that it brings to real-world use cases. We must thus identify promising application areas and develop convincing use cases for SMC, especially those that are non-zero-sum and present clear incentives for the adoption of SMC to all involved parties. In this thesis, such concrete use cases thus form the basis and the starting point of all our contributions. This approach follows the rationale that once concrete use cases are found, they motivate and narrow down further requirements for functionality, efficiency, and customization of SMC protocols.

In consequence, we choose a use case-driven research methodology for each of our contributions: Starting from the identified technological and societal trends for digital security and privacy (T1 – T4), our first step is always to identify promising problem fields and compelling use cases for SMC. Subsequently, we aim to demonstrate the real-world feasibility of the respective use case in a prototypical implementation and deployment. To this end, we analyze the technical feasibility regarding the state of the art and improve on the unveiled deficiencies with regard to functionality (Q1) and efficiency (Q2) within the context of the respective use case. We then look beyond traditional deployments and operations of SMC and show that our contributions can be tailored to the challenging requirements of novel emerging application scenarios in constrained environments (Q3). While the concrete use cases allow us to validate the real-world applicability of our contributions, we also deliberately abstract our contributions from them in order to show how our contributions benefit the field and adoption of SMC in general.


1.3 Contributions of this Thesis

In this thesis, we develop three main contributions (C1 – C3) that tackle the raised research questions. A mapping from our contributions to these questions is illustrated in Figure 1.2. Following the outlined use case-driven research methodology, each contribution targets different problem fields of digital security and privacy and stands for its own. We thus present for each contribution a dedicated motivation and problem analysis, a solution design and prototypical implementation, as well as a thorough evaluation and discussion of security and privacy properties. We now briefly summarize each contribution.

C1: Decentralized Mixing of Digital Currencies

Due to their promise of financial privacy, Bitcoin and other decentralized digital currencies are attractive alternatives to the centralized banking system and are even recognized as means for citizens to exercise their right for informational self-determination [Ger17] (T4). However, it has been shown users can be de-anonymized through various attacks [RH11, MPJ13, KKM14, BP15]. In response, several commercial services [Bit14a] offer to reestablish financial privacy. Their centralized design, however, makes them vulnerable to leaks, hacks and theft, as well as forced access by third parties [Her14, McM14, Bit16, Bit17b] (T3).

With our first contribution, we thus propose CoinParty [ZGH+15, ZMH16], an efficient and decentralized mixing service secured through SMC that reestablishes financial privacy in Bitcoin and related cryptocurrencies. Compared to previous mixes [Bit14a, Max13a, BNM+14, RMSK14], basing CoinParty on SMC offers significant benefits to users and service providers alike: Users benefit from stronger and extended privacy guarantees and lower costs while providers can rigorously protect themselves against hacks, leaks, and third party access that have caused significant financial loss in many Bitcoin-related ven-
1. Introduction

For the practical realization of CoinParty, we propose a novel combination of decryption mixnets with threshold signatures and improve existing techniques (Q1). Our newly developed functionality is useful beyond the scope of our immediate use case, e.g., for securing digital wallets [GBF+14]. We show that CoinParty can be deployed without any Trusted Third Party (TTP), is easy to access, and scales to large numbers of users.

In this first contribution, the main challenge consists in extending SMC functionality (Q1) and efficiency is only a secondary goal. Still, we show that CoinParty scales better than prior works and is efficient for realistic problem instances. Since the treated use case of mixing digital currencies, evidently, does not motivate further efficiency improvements (Q2), we move on to a different problem field, machine learning, that challenges us with much higher efficiency requirements – to meet these is subject to our second contribution.

C2: Privacy-preserving Pattern Recognition and Machine Learning

Pattern recognition and machine learning (PRML) have become indispensable tools in various application areas, especially in mobile services (T1), and are at the basis of data-driven business models and research (T2). However, we observe emerging applications and services in which the application of PRML is hindered because interacting parties are not able to share their data or models due to applicable data protection legislation or due to concerns over intellectual property and privacy. Such examples and settings are found, e.g., in biometrics [SSW09], location services [ZVHW14], or speech processing [PRRS13].

As our second major contribution, we present SHIELD [ZVHW14,ZMR+17], a framework for PRML which overcomes the identified problems using SMC. We realize secure variants of Naive Bayes, Hyperplane, and Artificial Neural Networks (ANNs) classifiers as well as the Forward and Viterbi on Hidden Markov Models (HMMs) – these are universally used classifiers and algorithms that are building blocks for many applications. We then apply and validate our secure designs in three concrete use cases: indoor localization, bioinformatics services, and spam recognition. The main challenge towards realizing secure PRML is improving the performance of non-integer arithmetic and generic function evaluation for which we significantly increase efficiency compared to the state of the art (Q2). We further demonstrate bandwidth-optimized building blocks [ZHH+15] and design secure outsourcing schemes in order to render the developed classifiers applicable in challenged environments (Q3).

Our evaluation shows that especially the secure outsourcing schemes present a promising approach to bring secure classification even to constrained devices and networks. This motivates us to further investigate the potential of secure outsourcing to bring SMC to more challenged operation and deployment scenarios (Q3), e.g., applications involving hundreds to thousands of mobile users that join and leave the computation dynamically over a constrained network.

C3: Secure Outsourced Set Intersection and Applications

A great variety of real-world applications require the computation of intersections of sensitive datasets, thus Private Set Intersection (PSI) protocols have hence become universal and important building blocks for securing these applications, e.g., in genomic testing [BBDC+11,DCFGT12]. PSI protocols often
need to be applied to large data sets [PSSZ15] and, as we additionally observe, in constrained or dynamically changing mobile environments (T1). We thus choose PSI as motivating use case and starting point of our third and final contribution addressing our third research question, customization of SMC protocols. Motivated by the preliminary results regarding this research question obtained in the context of our second contribution, we fully focus on outsourcing execution of SMC to cloud environments (T3).

As a first contribution, we present five outsourcing schemes for generic PSI protocols that cover a wide variety of scenarios, i.e., involving single, two, or multiple data holders who outsource to a single, two, or multiple cloud peers. Three of these schemes were published as part of our TraceMixer [ZHBW17] and BLOOM systems [ZPH+17]. For this thesis, we extended their evaluation and discussion and complement them with two further approaches to comprehensively cover all envisioned usage scenarios. The evaluation of these schemes show qualitative (in terms of security, composability, and flexibility) as well as quantitative (in terms of runtime and communication overheads) advantages over the state of the art. TraceMixer and BLOOM are complete systems for privacy preserving crowd sensing and genetic disease testing, respectively. Since they build on and extend our outsourced PSI protocols, we present these systems as comprehensive case studies for the real-world applicability of our outsourcing schemes for PSI. They should, however, be viewed as independent contributions as they significantly extend on the bare PSI outsourcing schemes.

With TraceMixer [ZHBW17] we present a novel location privacy protection mechanism tailored to the special requirements encountered in crowd sensing. We show how SMC allows designing TraceMixer in a trustless fashion that provides users with strong privacy guarantees by establishing k-anonymity [Swe02] on location trajectories in an oblivious manner using PSI. The scenario in focus of this case study, crowd-sensing, includes large numbers of mobile users that dynamically join and leave, participate with their smart phones, and communicate over constrained networks. We validate the applicability of TraceMixer to constrained mobile deployments (Q3) by mounting an actual crowd sensing campaign for the creation of high-precision altitude profiles.

The second case-study presents the BLOOM system [ZPH+17] for outsourcing tests for genetic variations of huge patient databases and a second separate contribution. The focus of this case study is to show how to customize SMC to allow single constraint users to harness the elastic storage and processing resources in the cloud. Securing BLOOM with SMC achieves strong security and privacy protection for the patients and at the same time unburdens service providers of legal, regulatory, or compliance requirements. BLOOM was submitted to the 2016 Secure Genome Analysis competition [iDa16] organized by the center for integrating Data for Analysis, Anonymization, and SHaring (iDASH). Submissions were judged by their performance (Q2) and the extent to which the constrained data holder is unburdened (Q3). BLOOM finished second among a total of eight submissions including solutions by Microsoft and IBM which validates our success in addressing our second and third research question.
1. Introduction

1.3.1 Attribution of our Contributions

The contributions presented in this thesis were developed in collaboration with students at COMSYS during their Bachelor’s and Master’s theses or in the context of lab projects. All major results were published with the help of the respective co-authors. In this section, we briefly credit students and co-authors with their work towards these contributions and their publication. If not stated otherwise, the core ideas and designs, the implementations and evaluations, as well as the final publication are attributed to the author of this thesis.

The motivation and problem analysis presented in this introduction are partly based on our survey of privacy threats in the Internet of Things [ZGMW14] and our comparative study of SMC frameworks [ZMHW15]. The survey was conducted at Philips Research Eindhoven and discussed with Oscar García-Morchon. The performance study was conducted jointly with Jan Metzke in his Bachelor’s thesis [Met14] according to the concept developed by the author of this thesis.

The initial motivation of our first contribution [ZGH+15, ZMH+16] was developed together with Fred Grossmann in the context of his lab project [Gro14]. Fred built and evaluated the first building block, the threshold signature scheme, following the design developed by the author of this thesis. Through valuable discussions with Martin Henze, the author of this thesis conceived and implemented the second important building block, the secure shuffle scheme. The author of this thesis then built and evaluated the first complete prototype [ZGH+15]. This prototype was conceptually and functionally improved together with Roman Matzutt in his Master’s thesis [Mat15]. The final system [ZMH+16] was evaluated with the help of Roman along his implementation.

For the second contribution, the author of this thesis designed, implemented, and evaluated the HMM algorithms in SHIELD [ZVHW14, ZMR+17] as well as the bandwidth optimized building blocks for constrained environments (BOMA) [ZHH+15]. According to the design and plan by the author of this thesis, Jan Metzke implemented the logsum building block of SHIELD and Jens Hiller helped to evaluate BOMA. The localization use case [ZVHW14] was developed with the help of Nicolai Viol. SHIELD is complemented by the joint work with Jan Metzke in the context of his Master’s thesis [Met17] during which he implemented and evaluated the additional Naive Bayes, Neural Network, and Hyperplane classifiers according to the designs by the author of this thesis (building and extending upon the approaches developed for the HMM algorithms published in [ZMR+17, ZVHW14]).

The first part of the third contribution, i.e., the general presentation of outsourced set intersection protocols, was extracted from the TraceMixer [ZHBW17] and BLOOM [ZPH+17] systems and extended by the author of this thesis by one additional scheme. TraceMixer [ZHBW17] itself represents a significant progression of early work with Jó Bitsch in the context of Nicolas Inden’s Master’s thesis [Ind14]. Following the design developed by the author of this thesis, Nicolas implemented and evaluated an early and unpublished prototype of a secure crowd-sensing system on top of which a use case was built based on Jó’s ideas. The main results of [ZHBW17], however, consist in the improved designs by the author of this thesis and were jointly developed, implemented, and evaluated with Jens Bavendiek in the context of his
Master’s thesis [Bav16]. Development of the BLOOM system [ZPH+17] was motivated by the 2016 Secure Genome Analysis Competition organized by the iDASH center [iDa16]. The original submission to the competition was developed jointly with a team of students, David Hellmans, Jan Pennekamp, Felix Schwinger, and Ike Kunze, who were supervised by Martin Henze, Jens Hiller, Roman Matzutt, and the author of this thesis. For the principal publication [ZPH+17], the author of this thesis significantly extended the submission to the competition by the design and implementation of a second complete approach that was evaluated jointly with Jan Pennekamp.

1.4 Outline

The main part of this thesis is structured as follows.

Chapter 2 provides an introduction to SMC and the relevant technical background for this thesis. We explain the underlying definitions and models of SMC and then present the state of the art protocols and technologies.

In Chapter 3, we demonstrate our first SMC use case, secure mixing networks to establish financial privacy in digital currencies. For its technical realization, we develop new functionality, e.g., a secure oblivious shuffle scheme, and customize existing technology, e.g., threshold signatures, such that the resulting system is easily accessible and usable even to mobile users.

Chapter 4 addresses the inherent security and privacy problems encountered in pattern recognition and machine learning. Here, we mainly face efficiency challenges and significantly improve upon state of the art techniques by proposing more efficient building blocks and classifier designs that we validate in different real-world use cases.

In Chapter 5, we motivate the computation of set intersections as important use case to examine how to customize SMC to non-traditional deployment and operation scenarios. Choosing outsourcing as the main solution approach, we first present a suite of generic outsourcing schemes for secure set intersection protocols and then present two case studies, privacy-preserving crowd sensing and genetic testing, that present independent contributions.

Chapter 6 concludes this thesis with a summary of our contributions and an outlook on open and future research challenges.
1. Introduction
Secure Multiparty Computation

In this chapter, we provide a concise overview of the topic of Secure Multiparty Computation (SMC). The approaches and results presented in this chapter build the technical foundation upon which we develop the protocols and systems that comprise the contributions of this thesis. Since our guiding goal is to make significant contributions towards the technical maturity and practical adoption of SMC, we confine the discussion of the theoretical foundation of SMC to a necessary minimum and focus on the important aspects of building and securing systems using SMC.

As a short introduction, we briefly trace the origins of SMC to distributed computing (Section 2.1). We then informally summarize the underlying security definitions and models (Section 2.2), and explain how generic secure protocols are constructed that establish powerful feasibility results for SMC in the defined security model (Section 2.3). Almost all actual instantiations and applications that form the current state of the art in SMC, as well as our own contributions, are based and extend upon these generic SMC protocols (Section 2.4). Finally, we summarize key aspects of the technical background provided in this chapter with regard to the motivation and research questions of this thesis (Section 2.5).

2.1 From Distributed Computing to SMC

The concept of SMC is best understood by looking at its origins, distributed computing. Distributed computing considers how a computation is carried out jointly by a set of distinct interconnected parties, i.e., in a distributed system. Real-world deployments of distributed systems and applications of distributed computing are ubiquitous today. Examples range from highly scalable databases, network file systems, and grid computing [OV11, KBM02] over routing and other algorithms in peer-to-peer networks [LCP+05], to novel computing or security paradigms such as blockchains [TS16] and distributed key management [RH03]. SMC considers exactly the same scenario as distributed computing, i.e., computing a joint function in a
distributerd system. The main difference between the two lies in the assumptions about what can possibly threaten the computation.

The transition from distributed computing to SMC is best motivated by the example of the Byzantine Generals Problem [LSP82], a classical problem in distributed computing. A group of Byzantine Generals besiege a city and need to agree on a coordinated attack or retreat; any disagreement will lead to the victory of the besieged army. The generals (distributed computers) can only communicate via messengers who may be captured and fail delivery (unreliable network links). The Byzantine Generals Problem illustrates a general challenge in distributed computing: how to ensure robustness of a computation when faced with inadvertent crashes and failures.

Different extensions and generalizations of the original problem have been proposed [BDM93], among them the following three important additions that greatly complicate the original problem: First, the Byzantine Generals decide to cast their votes privately, e.g., to avoid a biased consensus or to prevent the besieged army from anticipating the decision. Second, one or more generals are traitorous and try to sabotage the consensus, e.g., by sending inconsistent votes to provoke false majorities. Finally, captured messengers can be turned to deliver messages forged by the besieged army, e.g., in order to stir disagreement among the generals.

In these extended scenarios, the threat level is raised from inadvertent failures to deliberate attacks on the privacy and correctness of the computation. This marks the transition from distributed computing to secure computations: SMC asks how a group of parties can privately, correctly, and timely compute a joint function even under deliberate attacks by internal or external parties. Here, privacy means that each party learns only the output of the computation and nothing else, correctness requires that the honest parties cannot be tricked into computing a wrong result, and timeliness means that the computation among the honest parties cannot be infinitely held up. In the example of the Byzantine Generals, privacy allows each General to cast his vote secretly while correctness and timeliness guarantee that the honest Generals eventually reach the same consensus.

At first, the Byzantine Generals problem may seem artificially constructed and overly simplified to be relevant in today’s distributed systems. To contradict this widely spread disbelief, the U.S. NASA has documented several real-world Byzantine failures and their consequences [Dri12]. While NASA focuses on the original threat model, also the discussed generalizations and complications of the original problem occur in practice. Application scenarios for SMC range from rather harmless examples such as interest matching [HCE11] and electronic auctions [NP89,BCD+09] to highly critical applications such as electronic voting [DJN10] or distributed certification authorities [ZSVR02]. Imaginably, SMC could even be used to cryptographically enforce the two-man rule for nuclear weapons launches.

In these examples, the requirement for privacy and correctness can be quite differently pronounced. Correctness and timeliness are much less crucial, e.g., in interest matching than in electronic voting; privacy of nuclear weapons launch codes, e.g., is much more important than the privacy of bids in an online auction. To model such different requirements and, equally important, to clearly define the limitations of SMC, we require rigorous definitions of the underlying models and security guarantees of SMC. These are subject to the next section.
2.2 Model and Security Definitions

The exact model and security definitions of SMC are quite involved and not at all intuitive. We refer the reader to Goldreich’s book [Gol04] for a comprehensive treatment of this topic or to Lindell and Pinkas [LP09] for an excellent overview. The detailed formal definitions are important for a theoretic treatment of SMC, e.g., to prove general feasibility, composition theorems, or impossibility results [Can00]. In this thesis, we focus on practical aspects of SMC, in contrast, and thus restrict ourselves to rather informal yet more intuitive models and security definitions.

We consider a scenario with \( m \geq 2 \) distinct parties \( P_1, ..., P_m \) where each party \( P_i \) holds a private input \( x_i \). Together, they aim to compute a joint function \( (y_1, ..., y_m) = F(x_1, ..., x_m) \) over their inputs such that each \( P_i \) obtains her individual output \( y_i \) and learns nothing about the private inputs and outputs of the other parties other than what is implied by its own input \( x_i \) and output \( y_i \).

We illustrate this definition by a short example [HLOI16, KW15]: As of 2015, a growing number of more than 1300 active satellites orbit the earth [HLOI16]. Although, collisions are real [LS09], satellite operators keep the precise orbits of their satellites private, i.e., the very data that is required to predict and prevent future collisions. Cast as an SMC problem, the satellite operators aim to compute whether their satellites collide (the function \( F(\cdot) \)) based on their precise orbital information (the private inputs \( x_i \)) such that each operator \( P_i \) only learns which of its satellites are on a collision course (the private output \( y_i \)) and nothing else.

As motivated in the previous section, the joint computation of \( F(\cdot) \) can come under attack by an external adversary who may even have compromised a subset of the parties \( P_1, ..., P_m \). In this scenario, the basic problem considered in the field of SMC is the design of protocols that securely compute \( F(\cdot) \). Of course, we still need to define what security actually means in this context. The following are intuitive requirements that any secure protocol for the computation of \( F(\cdot) \) should fulfill [LP09]:

**Privacy:** We assume that all inputs are private to the respective owners and that the parties do not trust each other or any third party. Thus, \( P_1, ..., P_m \) must compute the joint function \( F(\cdot) \) without disclosing anything to anyone about their own inputs other than what is implied in the respective outputs \( y_i \) that each party \( P_i \) receives. In electronic voting, e.g., each party learns the outcome of the election but not the individual casted votes.

**Correctness:** If the computation succeeds, then each honest party is guaranteed to receive the correct output. In other words, dishonest parties must not be able to trick honest parties into computing a wrong result or bias the result other than through their choice of inputs. Continuing the example of electronic voting, this means that dishonest parties cannot influence the outcome of the election other than by casting their own vote according to their preferences.

**Timeliness:** The computation cannot be disrupted or halted by dishonest parties, i.e., honest parties eventually succeed in computing the function \( F(\cdot) \). For example, dishonest parties cannot stop the honest ones from determining a winner of the electronic vote.
These requirements are intuitive and a reasonable attempt to define security in SMC in a constructive manner. There is no guarantee, however, that they are complete -- we could have missed an important security requirement. We thus set out for a comprehensive definition of security that precludes any possible attack to the computation. To this end, we consider an ideal world where an external Trusted Third Party (TTP) exists that is incorruptible and thus mutually trusted by all participating parties. In this world, it is trivial to compute $F(\cdot)$ securely: Each party $P_i$ simply sends her private input $x_i$ to the trusted party (over a perfectly secure communication channel) who computes $(y_1, \ldots, y_m) = F(x_1, \ldots, x_m)$ on their behalf and passes each individual results back to the prescribed recipient.

In the real world, no trusted and incorruptible third party exists and the involved parties must hence compute the desired result in a distributed protocol without the help of a TTP. So, how does the ideal model enable a rigorous definition of security for real-world distributed computation protocols? We define a real protocol carried out by $P_1, \ldots, P_m$ for computing $F(\cdot)$ to be secure, if any successful attack on the protocol execution in the real world is also successful in the ideal world protocol execution. Since successful attacks are not possible in the ideal world (all computations are carried out by the incorruptible TTP), this definition implies that no attacks are possible in a real-world protocol execution either. Indeed, this security definition encompasses all of our three initial requirements, i.e., privacy, correctness, and timeliness. Privacy is guaranteed because the only information received by any party is its respective output. Parties hence cannot learn anything about any other party’s input other than what is implied in the received output. Correctness and timeliness both hold since the third party is incorruptible and will hence always correctly compute $F(\cdot)$ and return back the results in time.

A protocol that satisfies the stated security definition is called an SMC protocol. To prove that a protocol is secure with regard to this definition, it suffices to show that the protocol successfully emulates the ideal world. Formally, this requires that in any possible real and ideal execution of the protocol the distributions of the inputs and outputs are indistinguishable, which is usually proved using the real/ideal simulation paradigm: One shows that the view any party $P_i$ has on the protocol execution can be efficiently simulated given only its input $x_i$ and output $y_i$. We again refer to [Gol04, LP09] for the complete formal definitions and details.

So far, we have only defined what attacks we must withstand in the real world (all that are not also possible in the ideal world). We have not yet discussed what powers a potential adversary has for his attack on the protocol execution in the real world. This is discussed in the next subsection.

2.2.1 Attacker Model

In the SMC setting, we assume that the adversary can corrupt a subset of the parties that set out to jointly carry out a distributed computation securely using an SMC protocol. Depending on the application context, different assumptions are made about the capabilities of the adversary. In the following, we discuss the main types of adversaries considered in the literature. They are categorized by i) their attack behavior, ii) their corruption strategy, and iii) their computational complexity.
2.2. Model and Security Definitions

Passive versus Active Attacker Behavior: An important trait of the attacker is whether his attacks are passive or active. A passive adversary correctly follows the protocol but may try to gain additional information about the parties’ private inputs by analyzing the protocol transcripts and internal state of any passively corrupted party. Passive adversaries are thus also often referred to as semi-honest or honest-but-curious [HL10]. The passive adversary is thus a threat to privacy but not to correctness or timeliness. In contrast, parties corrupted by an active adversary are not bound by the protocol specifications. They may actively deviate from the protocol and cheat in arbitrary ways, e.g., to crash the computation, to infer the private inputs of others, or to forge the outcome. This setting is also referred to as the malicious attacker model.

Security against active adversaries is generally preferable, since it prevents any attack under worst case assumptions. However, security in this model is hard to achieve and resulting protocols often have high overheads. In comparison, the passive attacker model, though being rather optimistic, has many realistic applications and advantages. It allows for much more efficient protocols and protects against insider and outsider attacks when parties are not actively cheating [DSZ15, PSSZ15]. The passive model is especially realistic in computations that involve only two parties where one party consumes a service provided by the other. A common assumption is then that the service provider has a strong interest to behave at least semi-honestly and provide the offered service correctly in order to guard its reputation and fulfill contractual obligations [KMRS14, ATD15]. Furthermore, protocols in the semi-honest model often serve as stepping stones towards protocols secure in the stronger malicious model [DSZ15].

Static versus Adaptive Corruption Strategy: Orthogonal to the attack behavior is the question according to which strategy the adversary may corrupt parties in the protocol execution [CDD01]: A static adversary corrupts a fixed set of parties before the start of the protocol execution that remain corrupted throughout; likewise, honest parties remain honest. In particular, the adversary cannot corrupt parties depending on his view on the protocol execution — the adversary is non-adaptive. An adaptive adversary, in contrast, is able to arbitrarily corrupt honest parties during and depending on the protocol execution that remain corrupted until the end.

Both strategies have realistic real-world scenarios. The static corruption strategy fits well with passive attackers who are honest-but-curious and do not actively attack, e.g., for fear of losing reputation [ATD15]. Similarly, an adaptive corruption strategy nicely complements the active attacker’s behavior who is likely to take additional measures based on any partial information gathered during the protocol, such as hacking additional parties [CDD01].

Polynomial-time versus Unbounded Complexity: The final differentiation of the adversary’s power is his complexity. The adversary is either allowed polynomial or an unbounded time for his attack [LP09]. Results for polynomial-time adversaries usually assume the hardness of certain problems, e.g., factoring large integers or computing discrete logarithms, which have not been proven but are widely accepted [Nao03]. In contrast, an adversary without any computational limits can easily break most cryptographic primitives that are ubiquitously
used to secure today’s networks and applications, e.g., symmetric ciphers such as the Advanced Encryption Standard (AES) or public key algorithms such as Rivest-Shamir-Adleman (RSA) encryption or the Diffie-Hellman key exchange.

In practice, only polynomial-time adversaries are considered a threat. The assumption of unbounded adversarial complexity, however, leads to an interesting class of protocols that provide security regardless of any current or future computing technologies (e.g., quantum computing is a real threat to results in the polynomial-time complexity class [Sho97, FR99]) and free from cryptographic assumptions such as the existence of trapdoor permutations [RS89] or hard-to-solve problems [Na03]. Such protocols provide information-theoretic security [BGW88, CCD88] instead of computational security [Yao86, GMW87]. It is however important to note that they assume private communication channels between honest parties which the adversary can neither eavesdrop upon nor interfere with. In practical settings such as the Internet, such communication channels are realized through cryptographic primitives that are only computationally secure, which then renders the overall protocols only computationally secure. However, protocols secure against unbounded adversaries are still of theoretic interests as they afford for powerful composition results [Can90].

The focus of our contributions in this thesis is on the design of practical secure protocols for compelling real-world use cases. For this reason, we usually work in the semi-honest model with computationally-bounded adversaries to maximize performance and only consider active security where semi-honest behavior cannot be warranted, e.g., when the joint computation involves monetary values [ZGH+15, ZMH+16]. We emphasize that all semi-honest protocols presented in the course of this thesis are built upon SMC techniques that can be augmented with security against malicious adversaries. Security in the malicious model, inevitably, comes at the cost of significantly increased overheads [HKE12, KSS12] that are a primary optimization goal in ongoing research [KOS16, Lin16].

2.2.2 Limitations of Security Definitions

The security definitions presented in the previous section might seem all-encompassing as they preclude any adversarial success under the chosen attacker model. It is, however, important to bear in mind the following two aspects and limitations – they are implied in the security definitions but often go unnoticed.

First, security definitions for SMC only ensure that a distributed protocol for the desired functionality \( F(\cdot) \) is as good as having an incorruptible third party compute it. Importantly, this does not guarantee that the computed result does not leak non-trivial information about the parties’ private inputs. A popular minimal example that demonstrates this aspect is a secure protocol for answering the question whether Alice and Bob love each other (i.e., a logical AND on two bits). In case the result is “yes”, both parties trivially learn the other party’s input even if the protocol to compute the answer is perfectly secure in the strongest of the presented attacker models. Another intuitive example are electronic auctions where each party learns an upper bound on all other parties’ private bids from the public result, the value of the bid that won the auction. Before any real-world application, we should thus ask
whether we should compute a given functionality, at all, and if the computed results must be additionally protected. The latter can be achieved by combining secure computations with orthogonal approaches from the domain of database privacy, e.g., with \( k \)-anonymity [Swe02] or Differential Privacy [DMNS06,Dwo08], which limit the amount of information that can be inferred from the released result.

The second aspect that needs to be highlighted is the distinction between secure and private function evaluation: In Secure Function Evaluation (SFE), the jointly computed functionality \( \mathcal{F}(\cdot) \) is publicly known, while in Private Function Evaluation (PFE) the function itself is confidential. Note that our security definition actually covers both cases since the evaluation of a confidential function \( \mathcal{F}_{\text{conf}}(\cdot) \) can be reduced to the secure evaluation of a generic function \( \mathcal{F}_{\text{gen}}(\cdot) \) that takes as input arbitrary functions (of a maximum size) and evaluates them. The latter approach is referred to as a universal circuit [Val76] and is practical only for very small problem instances compared to the problem sizes that can be handled in SFE [SHS15,KS16].

Given the orders of magnitude higher overheads of PFE, we restrict ourselves to the case of SFE, which is also a better fit for real-world scenarios where computed functions and algorithms are usually established public knowledge and only the inputs require protection. Open algorithms are, furthermore, an important transparency measure that is especially important when dealing with private data, i.e., inputting private data into proprietary algorithms that have not been vetted can have severe consequences such as misdiagnosis of genetic diseases [Wil13]. A middle course between SFE and PFE can be taken by parameterizing the publicly known functionality and taking the actual parameters as private inputs, e.g., in machine learning where classification algorithms and the structure of classification models are publicly known but the concrete model may be the secret input of one party and the classified data record the secret input of the other (cf. Section 4).

Despite these two practical limitations, the presented definitions and attacker models are clearly a very rigorous and restrictive approach to security and one may ask whether secure protocols under these definitions are possible, at all. The short answer is yes, any distributed computation can be executed securely even in the strongest of the adversarial models discussed above. The long answer is given in the next section.

### 2.3 Generic Secure Protocols

The feasibility of SMC has already been proven in the late 1980s through the construction of generic protocols that allow securely executing any distributed computation [Yao86,GMW87]. Notably, these results hold even in the strongest of the adversarial models discussed above, i.e., against computationally unbounded and malicious attackers. At first, these general purpose SMC protocols were far from practical and only of theoretical interest. Three decades later, the early generic constructions are still at the core of most state of the art SMC protocols but have been tremendously improved.

We now briefly outline those constructions of generic secure protocols that are relevant for this thesis and refer the reader to [PGFW14,ABPP16] for a more comprehensive and detailed overview and classification of SMC protocols. Common to
all generic secure protocols is that they first represent the desired functionality abstractly through a combinational circuit, defined either over Boolean or arithmetic operations. Each individual gate in the circuit is then evaluated collaboratively by the protocol parties using an individual secure subprotocol for each type of gate.

Figure 2.1 Schematic view of 1-2-OT: The sender has two secret $s_0$ and $s_1$, the receiver inputs a choice bit $r$ and obtains $s_r$ from the sender. The OT protocol guarantees that the sender does not learn the choice $r$ and the receiver learns nothing about $s_{1-r}$.

Note that it is enough to define secure subprotocols for a functionally complete subset of gates, e.g., AND ($\land$) and XOR ($\oplus$) for Boolean or addition (+) and multiplication ($\times$) for arithmetic circuits. Furthermore, for any polynomial-time function a corresponding polynomial size arithmetic or Boolean circuit can be efficiently constructed [KSS10]. Thus, secure computation over Boolean and arithmetic circuit representations are equivalent regarding the functional expressiveness. Especially the construction of Boolean circuits has been automated and heavily optimized, drawing on existing software for logic synthesis from the domain of hardware circuit design [MNPS04, HFKV12, SHS15, DDK15]. In contrast, constructing arithmetic circuits requires more handwork, but can result in much smaller circuits as arithmetic operations over $l$ bit integers can be expressed in single operations while typically requiring $\Theta(l)$ gates in a Boolean circuit [KSS10].

In the following, we present generic secure protocols constructed over Boolean circuits (Section 2.3.2) and arithmetic circuits (Section 2.3.3). For both approaches, Oblivious Transfer (OT) is an important building block introduced first (Section 2.3.1). Note that the related literature differentiates between the two-party case ($m=2$) and the multi-party case ($m>2$), since these two scenarios have lead to very different protocol constructions. From hereon, we explicitly distinguish these two cases where necessary and refer to them as Secure Two-Party Computation (STC) and Secure Multiparty Computation (SMC).

### 2.3.1 Oblivious Transfer

Oblivious Transfer (OT) is a two-party protocol originally introduced by Rabin in 1981 [Rab05] that allows one party, the receiver, to learn exactly one of multiple secrets held by the other party, the sender, who learns nothing about the receiver’s choice. In other words, OT guarantees that the sender remains oblivious to the receiver’s choice and that the receiver remains oblivious of all the sender’s secrets except for the choice. OT is an essential building block for generic STC protocols, e.g., it is used in Yao’s Garbled Circuits (GCs) protocol (Section 2.3.2.1) to obliviously share encryptions of the input bits or in the Goldreich-Micali-Wigderson (GMW) protocol (Section 2.3.2.2) to evaluate AND gates.
2.3. Generic Secure Protocols

In the most basic case, 1-2-OT, the sender $S$ holds two secret bits $s_0$ and $s_1$ while the receiver $R$ holds a choice bit $r \in \{0, 1\}$ as illustrated in Figure 2.1. Running the 1-2-OT protocol, $R$ obtains exactly $s_r$ and learns nothing about $s_{1-r}$ while $S$ learns nothing about the choice $r$.

We illustrate this definition and the usefulness of OT by constructing a first simple STC protocol that allows two parties, Alice and Bob, to securely determine whether they love each other (i.e., the toy example introduced in Section 2.2.2 to discuss the limitations of SMC): Alice and Bob engage in an OT where Alice acts as receiver and inputs choice 0 if she does not love Bob and choice 1 if she does. Bob acts as sender and inputs two secrets of which Alice learns exactly one: The first secret is always 0 indicating that Bob does not love Alice; the second secret expresses his true feelings in a single bit 0 or 1. It is easy to see that the result is 1 only if both parties love each other and 0 in all other cases such that no party is humiliated by the overtly unreturned love.

The basic 1-2-OT definition can be generalized in three dimensions, the bitlength $l$ and the number $n$ of secrets held by $S$ as well as the total number $m$ of parallel executions of the OT protocol between $R$ and $S$. In the most general form, 1-\(n\)-OT\(_m^l\), $S$ thus holds a batch of $m$ distinct $n$ tuples of $l$-bit strings and $R$ independently chooses exactly one string from each $n$ tuple, while $S$ learns nothing about $R$’s choices. Formally, $S$ holds $(s_{1,1}, ..., s_{1,n}), ..., (s_{m,1}, ..., s_{m,n})$ with $s_{i,j} \in \{0, 1\}^l$ and $R$ holds $m$ choices $r_1, ..., r_m \in \{1, ..., n\}$ and obtains the strings $s_{i,r_i}, 1 \leq i \leq m$ while $S$ has no output. We now discuss how to efficiently instantiate OT.

Extending Oblivious Transfer

OT can be realized using public key cryptography and the currently most efficient 1-2-OT protocol of [NP01,NP05] has a cost of approximately three modular exponentiations (approximately three times the costs of an RSA encryption). While this is practical for a moderate number of OTs, the costs become prohibitive when a large number of OTs has to be computed, e.g., millions to billions as is the case in recent applications of STC [BCF+14,PSSZ15]. The basic idea to solve this problem is very similar, e.g., to efficient encryption of emails or Internet traffic where only a short symmetric key is exchanged using expensive asymmetric cryptography and the payload itself is encrypted using cheap symmetric key primitives only (e.g., the AES). In the same fashion, the OT extension protocols proposed in [Bea96,IKNP03,ALSZ13] extend few short OTs (the base OTs) into many long OTs (the extended OTs) using only a small number of cheap symmetric cryptography and one-time-pad operations.

The concept of OT extensions is illustrated in Figure 2.2. The first step is to run a small number of short base OTs (dark gray box, top left), e.g., usually employing the most efficient available protocol for 1-2-OT [NP01,ALSZ13]. In an analogy to the example of email encryption, this step corresponds to running a key agreement protocol based on asymmetric cryptography. The exact number and size of the base OTs depends on the desired symmetric security level $t$ which measures the strength of the involved cryptographic primitives – a security level of $t$ bits means that an attacker has to perform on average $2^t$ operations to break security. We summarize typical security levels used in the literature and in this thesis together with further notation in Section 2.3.4.
The short base OTs over $t$ bit strings can be securely extended horizontally to OTs over long strings using only cheap symmetric cryptography. The small number of $t$ OTs can be extended vertically to a large number of $m > t$ OTs using one-time-pad operations. Remarkably, these OT extensions can be combined to extend OT in both dimensions using only cheap symmetric cryptography and one-time-pad operations. The currently most efficient protocol for extending OTs in both number and length is given in [ALSZ13]. In our email encryption example, the extension step roughly corresponds to using a single short symmetric key for the encryption of multiple messages of arbitrary length.

The most general primitive, $1-n\cdot OT^m_l$ (not shown in the figure), is efficiently instantiated by reducing it to a batch of $m \cdot \log_2(n)$ parallel 1-2-OT, which can then be computed using the outlined OT extensions.

All protocols introduced above offer security against computationally bounded semi-honest adversaries. A state of the art OT extension protocol with stronger security against malicious adversaries is given in [ALSZ13]. In this thesis, we exclusively use semi-honest OT extensions due to their superior performance, but emphasize that this building block can be easily replaced in all our contributions by the maliciously secure variant if stronger security is required.

### 2.3.2 Secure Computation of Boolean Circuits

We now discuss the first flavor of generic secure protocols, those defined over Boolean circuits. This class of secure computations is predominantly represented by Yao’s Garbled Circuits [Yao86] and the Goldreich-Micali-Wigderson protocol [GMW87]. Common to these and other approaches is that the desired function is in a first step represented as a combinatorial Boolean circuit (which is possible since Boolean logic is functionally complete). The difference between the approaches in this class of secure protocols is how they encrypt inputs and then securely evaluate the individual gates of the circuit on the encrypted inputs.
2.3. Generic Secure Protocols

2.3.2.1 Yao’s Garbled Circuits

In 1986, Yao proposed his Garbled Circuits as the first generic STC protocol [Yao86] to solve his famous Millionaire’s Problem introduced four years earlier [Yao82]. The core idea is to let one party encrypt the Boolean circuit that represents the joint function on all possible inputs such that the other party can evaluate under encryption exactly the path through the circuit that corresponds to the parties’ inputs. Yao’s original protocol is, thus, a two-party protocol, but an extension to the multi-party case has been proposed in [BDNP08]. However, multi-party GCs have received less attention since more efficient secret sharing-based protocols exist for multi-party computation that we will discuss in Section 2.3.3.

Overview: We present Yao’s original two-party version that runs in the following three phases. 1) Garbling the circuit: The first party, the garbler $G$, garbles the Boolean circuit $F_{\text{Bool}}(\cdot)$ representing the joint function $F(\cdot)$, its own input $x$, and all possible inputs of the second party, the evaluator $E$. $G$ then transmits the garbled circuit $F_{\text{Garb}}(\cdot)$ as well as his own garbled input $\tilde{x}$ to $E$, and $E$ receives her own garbled input $\tilde{y}$ from $G$ via OT. 2) Evaluating the garbled circuit: $E$ locally evaluates the garbled circuit $F_{\text{Garb}}(\cdot)$ using the garbled inputs $\tilde{x}$ and $\tilde{y}$ and obtains the garbled results $\tilde{z}$ – the garbling prevents that she learns anything about the inputs, the intermediate values, or the result. 3) Reconstructing the results: Depending on who should learn the result, either $E$ sends the garbled result $\tilde{z}$ to $G$ or $G$ provides the output mapping of garbled to clear text results to $E$, which $E$ uses to ungarble $\tilde{z}$.

We now explain each phase in more detail in the following. As a simple illustrative example, we consider the evaluation of a minimal Boolean circuit that consists of a single logical AND ($\land$) gate, i.e., the inputs $x, y$ and the output $z$ are single bits.

Garbling the circuit: Given the Boolean circuit representation $F_{\text{Bool}}(\cdot)$, the first party, the garbler $G$, garbles each gate $g \in F_{\text{Bool}}(\cdot)$ as follows. We denote the gate’s two input wires by $W_a$ and $W_b$ and the output wire by $W_c$ (Figure 2.3(a)). Each gate is then expressed by its truth table $T_g$ (Figure 2.3(b)). $G$ now assigns to each wire $W_i \in \{W_a, W_b, W_c\}$ of the gate two random $t$ bit symmetric keys $k_i^0, k_i^1 \in \{0, 1\}^t$; one key represents the input 0 and the other the input 1. These keys are referred to as garbled inputs and outputs. Now, $G$ additionally

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>$g_a$</th>
<th>$g_b$</th>
<th>$g_c$</th>
<th>$T_g$</th>
<th>$\tilde{T}_g$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$E_{\text{G}}(E_{\text{G}}(0))$</td>
<td>$\tilde{x}$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>$E_{\text{G}}(E_{\text{G}}(1))$</td>
<td>$\tilde{y}$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>$E_{\text{G}}(E_{\text{G}}(1))$</td>
<td>$\tilde{y}$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>$E_{\text{G}}(E_{\text{G}}(1))$</td>
<td>$\tilde{y}$</td>
</tr>
</tbody>
</table>

Figure 2.3 Garbling one circuit gate by the example of a logical AND: The gate $g \in F_{\text{Bool}}(\cdot)$ (a) is first expressed as truth table $T_g$ (b). The truth table is then garbled by encrypting it (c) and stripping the inputs and permuting the remaining encrypted and garbled outputs (d).
encrypts each garbled output value $k^0_i$ and $k^1_i$ using the corresponding two garbled inputs from the truth table as keys (Figure 2.3(c)). For example, the garbled inputs corresponding to the third row of $T_2$ are $k^2_1$ and $k^2_0$, so the corresponding garbled output $k^2_0$ is encrypted by $E_2((E_0(k^2_0))$ using an efficient semantically secure encryption scheme $E$ such as AES (the order of encryption is well defined by the enumeration of the wires). Finally, the garbled inputs are stripped and the remaining encrypted and garbled outputs are permuted so that their position does not leak the original values (Figure 2.3(d)).

In this fashion, $G$ garbles all gates of the circuit, i.e., using the garbled outputs of one gate as keys to garble directly connected subsequent gates, and sends the stripped and permuted tables $T^g$ for each gate $g \in F_{\text{Boo}}(\cdot)$ to $E$. $G$ also provides its own garbled input $\tilde{x}$ to $E$ ($\tilde{x} = k^2_2$, in our toy example). Note that $E$ cannot learn $G$’s input $x$ from $\tilde{x}$ since the received keys are completely random to $E$, i.e., she cannot distinguish $k^0_0$ from $k^1_0$ in our example.

**Evaluating the garbled circuit:** The evaluator $E$ receives the complete garbled circuit $F_{\text{Boo}}(\cdot)$ as well as $G$’s garbled input $\tilde{x}$. To be able to decrypt the garbled circuit gate by gate, $E$ also needs its own garbled input $\tilde{y}$. However, $G$ must not learn $E$’s private input $y$ and, hence, cannot select the right keys $\tilde{y}$ to send them to $E$ directly. Furthermore, $G$ cannot send all possible keys (i.e., $k^0_i$ and $k^1_i$ in our example) since this would allow $E$ to decrypt more than one path through the garbled circuit which breaks $G$’s privacy. The solution to this problem is for $G$ and $E$ to engage as sender and receiver in one 1-2-OT protocol run per input bit of $E$, e.g., $G$ inputs $k^0_i$ and $k^1_i$ and $E$ inputs the choice bit $y$ and receives exactly $k^y_i$ while $G$ remains oblivious of $E$’s choice $y$.

Using $k^y_2$ and $k^0_2$, $E$ tries to decrypt each entry in $T^g$. $E$ can decrypt only one row correctly and obtains exactly the garbled output value corresponding to the inputs $x$ and $y$, i.e., $k_i^x \land y$ in our example. The decrypted but still garbled output $\tilde{z} = k_i^x \land y$ is then used to decrypt subsequent garbled gates.

Note that $E$ cannot learn the real output $x \land y$ from the decrypted but garbled output $k_i^x \land y$ since keys are completely random to $E$. Remarkably, $E$ still correctly evaluates the gate $g$ without knowing what was computed. To evaluate the whole circuit, $E$ iteratively decrypts each gate in $F_{\text{Boo}}(\cdot)$ using the decrypted garbled outputs as keys to decrypt the connected subsequent gates.

**Reconstructing the results:** Finally, $E$ obtains the garbled results, i.e., the garbled outputs of all gates whose output wires are not the input wires of other gates. In our example (cf. Figure 2.3), the final garbled result is $\tilde{z} = F_{\text{Boo}}(\tilde{x}, \tilde{y})$. In order to release the result to $E$, or $G$, or to both parties, we proceed as follows: If only $E$ should learn the result, $G$ sends all keys corresponding to output wires of the circuit to $E$ together with the mapping to the real outputs, e.g., $k^0_0 \mapsto 0$ and $k^1_1 \mapsto 1$ in our example (cf. Figure 2.3(c)). $E$ then just needs to look up the garbled output $\tilde{z}$ in the output mapping to obtain the plaintext output. If only $G$ should learn the result, $E$ just sends $\tilde{z}$ to $G$ who then obtains the output in the same way. If both should learn the result, $E$ sends $\tilde{z}$ to $G$ who ungarbles the output and additionally sends it to $E$ in clear.

---

1. The employed encryption scheme must allow to determine incorrect decryptions, which can be achieved for efficient block ciphers such as AES without additional overheads [BHKR13], but is left out for the sake of brevity.
State of the art: The original version of Yao’s protocol [Yao86] is far from practical [MNPS04], but its performance has been greatly improved over the years. An impressive line of research (point-and-permute [BMR90, MNPS04], row-reduction [NPS99], free XOR [KSS09], and half-gates [ZRE15]) has reduced the required number of cryptographic operations per gate from 13 in the original protocol down to 6 per AND and 0 per XOR gate. Similarly, these techniques reduce the communication overheads from 4 keys per circuit gate down to 2 keys per AND gate and, again, 0 per XOR gates. The authors of the latest optimization, half-gates [ZRE15], argue that a lower bound has been reached. Thus, an orthogonal line of research has been the reduction of the size of the Boolean circuit representation and its optimization for the free XOR technique [KSS09]. To this end, size-efficient circuit building blocks have been designed by hand [KSS09] or obtained by adopting logic synthesis from the field of hardware design [SHS +15, DDK +15] – note that finding minimal Boolean circuits is generally an NP hard problem [KC00]. Other proposals optimize the speed of the garbling primitives or the execution backend, e.g., garbling using fixed-key AES [BHKR13] or pipelining [HEKM11]. Finally, to achieve security against malicious adversaries, among others, efficient cut-and-choose techniques have been developed [LP07, KSS12, S +13, Lin16]. Yao’s protocol requires only three rounds of communication independent of the computed function and provided inputs, which renders it susceptible to network latency between the garbler \( G \) and evaluator \( E \). Also, large fractions of the overall overhead can be precomputed by \( G \) without knowing any prior interaction or knowledge of the other party \( E \) and her inputs. However, the evaluation of the garbled circuit in the online phase requires a number of (cryptographic) operations linear in the size of the circuit. In the next section, we present a second approach to securely evaluate Boolean circuits that realizes a different performance trade-off.

2.3.2.2 The Goldreich-Micali-Wigderson Protocol

Shortly after Yao’s initial proposal, Goldreich, Micali, and Wigderson presented their approach to generic secure computations [GMW87] which applies to both the two-party and multi-party setting. In the following, we concentrate on the two-party version of the GMW protocol for the sake of simplicity (the multi-party case is conceptually similar). Just like Yao’s GC protocol, the GMW protocol is defined over Boolean circuits. However, instead of garbling the circuit, the core idea is to share inputs bitwise between all parties such that single shares do not reveal the inputs but still allow all parties to collaboratively evaluate the Boolean circuit.

Overview: The GMW protocol proceeds in the following three phases that are executed by both parties, \( P_1 \) and \( P \) (we use generic names as parties do not have special roles). 1) Input sharing: Each party computes an XOR-sharing of its own inputs and distributes shares to the other party. 2) Evaluating the Boolean circuit: Using the secret shares of the inputs, both parties collaboratively evaluate each gate of the plain Boolean circuit (i.e., other than in Yao’s GCs, the circuit is not garbled or modified in any way). 3) Reconstructing the result: Depending on who should learn the final result, the two parties exchange the individual secret shares of the result and obtain the clear text by XOR-ing them.

2Note that the combination of AND (\( \wedge \)) and XOR (\( \oplus \)) is functionally complete.
We now explain each phase in more detail. Since the steps and tasks for both parties are exactly the same, it suffices to describe the part of $P_1$. We assume single bit values (as in our example of Figure 2.3) to simplify notation. In the general case of $t$-bit values, each operation is simply performed $t$ times in parallel.

**Sharing the inputs:** Let $P_1$ hold input $x \in \{0, 1\}$. $P_1$ draws a random bit $r \in \{0, 1\}$ and masks her input $x \oplus r$. The tuple $(x) = (r, x \oplus r)$ is referred to as a 2-out-of-2 Boolean sharing\(^3\) of $x$, since $r \oplus (x \oplus r) = x$. Note that one individual share, i.e., $r$ or $x \oplus r$, reveals nothing about the shared value $x$. Thus, $P_1$ can safely provide $r$ to $P_2$ and keep $x \oplus r$ as her own share. We denote the individual shares by $(x)_1$ and $(x)_2$. In parallel, $P_1$ receives and stores the share of $P_2$’s input $y$.

**Evaluating the Boolean circuit:** We have to show that $P_1$ and $P_2$ can together evaluate the Boolean circuit. To this end, it suffices to show how they evaluate a functionally complete subset of logical gates. Given the many optimizations developed for circuits consisting of AND and XOR gates in the context of Yao’s GC protocol, it is natural to select these two types of gates also for GMW.

The basic idea is that both parties use only their shares to evaluate a gate and obtain any intermediate result, i.e., the output of any gate, again in the same shared form such that they can proceed to evaluate connected subsequent gates in the same fashion.

**XOR gates:** As in GCs, evaluating XOR gates comes virtually for free, since both parties can compute the desired share locally in just one XOR operation: Note that $x \oplus y = ((x)_1 \oplus (x)_2) \oplus ((y)_1 \oplus (y)_2) = ((x)_1 \oplus (y)_1) \oplus ((x)_2 \oplus (y)_2)$. Thus, $(x \oplus y) = ((x)_1 \oplus (y)_1, (x)_2 \oplus (y)_2)$ is a correct sharing of $x \oplus y$ and can be computed locally by each party $P_i$ from its individual shares $(x)_i$ and $(y)_i$.

**AND gates:** Evaluating AND gates using shares is more difficult and requires interaction. $P_1$ and $P_2$ engage in one run of 1-4-OT where $P_1$ acts as receiver and inputs the choice $(z)_1 \in (0, 1)^2$ (a bitstring of length 2). Note that $P_1$ randomly draws her own share $(z)_2 \in \{0, 1\}$, then acts as sender and inputs four possible shares $(z)_1, = (z)_2 \oplus (((z)_1 \oplus (z)_2) \land ((y)_1 \oplus (y)_2))$, i.e., one for each of the four possible choices $i = (x)_1 (y)_1 \in \{0, 1\}^2$. After the OT run, $P_1$ holds $(z)_1$ and $P_2$ holds $(z)_2$ with $(z)_1 \oplus (z)_2 = x \land y$. Note that $P_1$ and $P_2$ may change roles of sender and receiver in the OT after the evaluation of each AND gate, e.g., to equally distribute the slightly asymmetrical loads of OT.

**Reconstructing the result:** Using the presented two techniques, both parties can jointly evaluate the whole circuit iteratively. Finally, each party obtains a share of the final result. If $P_1$ should learn the final result in clear, $P_2$ sends her share and $P_1$ simply recombines the final result (and vice versa).

**State of the art:** Surprisingly, the first implementation of the GMW protocol was proposed only in 2012 [CHK+12], eight years after GCs had been implemented in [MNPS04]. Since GMW requires one round of interaction per sequential AND in

---

\(^3\)Not to be confused with our notation for OT, “2-out-of-2” in the context of secret sharing means that it takes two distinct shares out of a total of two shares to reconstruct the shared secret.
the circuit, it was believed to be much less efficient than Yao’s constant rounds GC protocol. In [SZ13], the authors heavily optimize the GMW protocol in the two party setting and show that GMW can indeed beat GCs performance-wise in networks with moderate latency. Further significant improvements are achieved through the use of multiplication triples [Bea91] together with specialized OT primitives [ALSZ13] to speed up the evaluation of AND gates. Furthermore, logic synthesis software can be adapted to optimize circuits for low multiplicative depth (the number of sequential AND gates) in order to minimize communication rounds [DDK+15]. Finally, the GMW protocol has been extended to provide security against malicious adversaries [NNOB12], which introduces significant additional overheads.

Comparing fully-optimized GMW to fully optimized GCs, GMW’s striking advantage is that all cryptographic operations (for the computation of Boolean multiplication triples using OT) and the majority of the traffic can be shifted into a precomputation phase [SZ13]. The online phase, in contrast, then involves only very cheap one-time-pad operations but still requires rounds of communication linear in the multiplicative depth of the circuit. A second major advantage is the reduced memory complexity from \( t \) bits per wire in the circuit in GCs to only a single bit per wire in GMW [SZ13].

### 2.3.3 Secure Computation of Arithmetic Circuits

Both Yao’s GCs and the GMW protocol require the bitwise computation of the joint functionality. Intuitively, this is an adequate approach to logical computations but seems overly costly for computations where inputs and intermediate values consist of multiple bits such as in arithmetic computations. In this section, we thus concentrate on the second flavor of secure protocols, those defined over arithmetic instead of Boolean circuits. The predominant approaches in this category are based on Linear Secret Sharing (LSS) or Homomorphic Encryption (HE).

#### 2.3.3.1 Linear Secret Sharing

The first secure protocol for arithmetic circuits was proposed by Ben-Or, Goldwasser, and Wigderson (BGW) in 1988 [BGW88]. The core idea is similar to the GMW protocol, i.e., each party shares inputs between all parties such that single shares do not reveal the inputs but still allow all parties to collaboratively evaluate the gates of the arithmetic circuit, i.e., additions and multiplications. Since the BGW protocol is defined over arithmetic circuits, it utilizes a secret sharing scheme defined over the integers instead of the bitwise XOR sharing of Boolean values used in GMW. The only difference, thus, lies in the underlying circuit representation and, consequently, how secrets are shared and individual gates are evaluated.

**Overview:** The desired functionality \( F(\cdot) \) is first expressed as an arithmetic circuit \( F_{\text{arith}}(\cdot) \) of addition and multiplication gates which is subsequently evaluated jointly by all parties \( P_1, \ldots, P_n \) in three phases similar to the GMW protocol: 1) **Sharing the inputs:** All parties compute shares of their private inputs and distribute these to the others 2) **Evaluating the arithmetic circuit:** Using the shared inputs, the parties collaboratively evaluate the arithmetic circuit gate by gate. 3) **Reconstructing the result:** Depending on who should learn the final result, the parties exchange the individual secret shares of the result and reconstruct the clear text from them.
As we will see in the following, SMC protocols based on \textit{LSS} require addition and multiplication to be defined over a finite ring, e.g., $\mathbb{Z}_q$. Some approaches even require a finite field such as $\mathbb{F}_p$, e.g., when inversion of elements is required. It is beneficial to choose a ring as underlying algebraic structure where possible as this increases the efficiency of the implementation. For example, arithmetic in the ring $\mathbb{Z}_q$ for $l = 8, 16, 32, 64$ is natively supported by compilers and processors through the \texttt{uint8_t, uint16_t, uint32_t, and uint64_t} data types. Operations on these data types are highly efficient since they correspond directly to operations on the processor’s registers of the respective widths. In contrast, arithmetic in finite fields usually requires special libraries. In the following, we thus note explicitly when a finite field is required; otherwise, a finite ring suffices.

The original BGW protocol [BGW88] applies only to the multi-party case but can be adopted to the two-party setting with some modifications. In this thesis, we employ it in both settings and thus describe it for both cases. As before, we discuss each phase in more detail, then present state of the art optimizations.

\textbf{Sharing the inputs:} In the first phase, any party $P_i$ with an input $x \in \mathbb{N}$ shares this input to the other parties using a \textit{LSS} scheme. Different schemes are used in the multi-party and two-party setting. In both cases, the interesting property is that no party learns anything from an individual share or even any “not-too-large” subset of shares in the multi-party case.

- In the multi-party case [BGW88], Shamir’s secret sharing [Sha79] is used to share a secret $x$ to $m \geq 3$ peers such that nothing can be learned about $x$ from any subset of less than $t \leq m$ shares, referred to as a $t$-out-of-$m$ secret sharing.\footnote{Denoting the recombination threshold by $t$ collides with our notation of the symmetric security level $t$ introduced earlier. We stick to this notation since it common in the literature and the two parameters have no relation to each other and the meaning can thus be distinguished by the context.} To share $x$, $P_i$ draws a random polynomial $f_x \in \mathbb{F}_p[X]$ of degree $t - 1$ with $f_x(0) = x$ and sends $\langle x \rangle_j = f_x(j)$ to $P_j \forall j = 1, \ldots, m$.

- In the two-party case [DSZ15], 2-out-of-2 additive secret sharing is used. To share $x \in \mathbb{Z}_2^l$, $P_1$ selects a random number $r \in \mathbb{R}$, sends $\langle x \rangle_2 = r$ to $P_2$ and keeps $\langle x \rangle_1 = x - r$ as its own share (analogous for $P_2$).

\textbf{Evaluating the arithmetic circuit:} As before, we show that the parties $P_1, \ldots, P_m$ can together evaluate the arithmetic circuit using the shares of the inputs. It suffices to provide subprotocols for addition and multiplication gates since these operations are functionally complete [KSS10].

\textit{Addition:} In both the multi- and two-party setting, addition gates can be computed locally since the used secret sharing techniques are additive, i.e., a party with two arithmetic sharings $\langle x \rangle$ and $\langle y \rangle$ can simply compute $\langle x \rangle + \langle y \rangle$ within the chosen ring or field to obtain a valid share $\langle x + y \rangle$. Note that multiplication by a public scalar $s$ is equivalent to executing $s - 1$ additions and can thus be computed locally by each party simply by multiplying $s$ with the share of $x$ in the given ring or finite field, i.e., $\langle s \cdot x \rangle = s \cdot \langle x \rangle$.

\textit{Multiplication:} Multiplication of two shared values $\langle x \rangle$ and $\langle y \rangle$ is more involved and requires interaction between the parties. Again, we require different protocols in the multi- and two-party setting.
2.3. Generic Secure Protocols

• In the multi-party case, multiplication requires three steps [BGW88]:
  First, each party locally multiplies its shares of the factors $x$ and $y$, i.e.,
  \[\langle xy\rangle_i = \langle x \rangle_i \cdot \langle y \rangle_i,\] which shares the product $xy$ over a polynomial of degree $2(t-1)$. Second, to reduce the degree back to $t-1$ (necessary for subsequent multiplications) each party reshares the local result $\langle xy\rangle_i$ to the other parties by sending $\langle\langle xy\rangle_i\rangle_j$ to $P_j$. Finally, each party recombines (as explained below) a share of the product $xy$ over a polynomial of degree $t-1$ from the double shared products $\langle\langle xy\rangle_i\rangle_j$, received from the other parties $P_j = 1..m$.

• In the two-party case, where we used additive arithmetic sharings, the multiplication can be expressed as follows:
  \[x \cdot y = (\langle x \rangle_1 + \langle x \rangle_2) \cdot (\langle y \rangle_1 + \langle y \rangle_2)\]
  \[= \langle x \rangle_1 \cdot \langle y \rangle_1 + \langle x \rangle_2 \cdot \langle y \rangle_2 + \text{interactive} \langle x \rangle_1 \cdot \langle y \rangle_2 + \text{interactive} \langle x \rangle_2 \cdot \langle y \rangle_1\]
  The first two terms can be computed locally by the two parties. In contrast, computing the mixed terms requires interactive protocols based on OT [Gil99] or HE [ABL+04] (HE will be discussed in the next subsection).

Reconstructing the result:
Using the presented techniques, the parties can jointly evaluate the whole arithmetic circuit gate-by-gate. Finally, each party thus obtains a Shamir or an additive share of the final result. The parties now send their individual shares to any party who should learn the result.

• To recombine a Shamir-shared value $x$ in the multiparty case, any party $P_i$ first needs to obtain at least $t$ shares and can then use Lagrange interpolation to reconstruct $x$ from the shares. The party computes
  \[x = f_x(0) = \sum_{i=1}^{t} \langle x \rangle_i \cdot \prod_{j \neq i}^{m} \lambda_{i,j}\]
  with $\lambda_{i,j} := \frac{i-j}{j-i}$ the Lagrange basis polynomial.

• Recombining an additive share in the two-party setting is done by adding both shares within the ring.

State of the art: Clearly, the bottleneck of the BGW protocol are the interactive multiplication steps which have thus been subject to optimization. An important approach is to precompute multiplication triples [DN07, DGKN09, KOS16]. Other approaches optimize for different levels of parallelism [BSMD10, ZSB13]. An important orthogonal line of optimizations investigates the efficient representation of higher level building blocks as arithmetic circuit [BSMD10, CDH10, CS10, ABS13]. Protocols and optimizations over additive secret sharing have been proposed in [ABL+04, PBS12, DSZ15]. Highly efficient protocols for additive secret sharing between three fixed parties have been proposed in [BLW08, KW15]. All of the discussed protocols and optimizations are only secure against semi-honest adversaries. Security against malicious adversaries can be achieved using verifiable secret sharing [RB89, Ped92, ALR11, ABB16] or a mix of different techniques [DIK10, KW15], but adds significant overheads.
As a fourth and final approach, we show how secure protocols can be built based on Homomorphic Encryption (HE) schemes. The defining property of all HE schemes is that the encryption function $\text{Enc}(\cdot): x \mapsto [x]$ defines a homomorphism between the plaintext domain $\mathcal{P}$ and the ciphertext domain $\mathcal{C}$ as illustrated in Figure 2.4 (and decryption $\text{Dec}(\cdot): [x] \mapsto x$, vice versa). This implies that there exists at least one operation $\ast$ (e.g., addition (+) in Figure 2.4) on the clear texts that maps to an operation $\odot$ ($\oplus$ in Figure 2.4) on the ciphertexts such that for each pair $x, y \in \mathcal{P}$ it holds that $[x] \odot [y] = [x \ast y]$. HE schemes come in three flavors:

**Partially Homomorphic Encryption (PHE)** allows the computation of a single arithmetic operation, addition or multiplication, on ciphertexts. E.g., the Paillier [Pa99, DJ01] and Damgård-Geisler-Krøigaard (DGK) [DGK07, DGK08] schemes are additive homomorphic, i.e., $[x + y] = [x] \odot [y] = [x] \cdot [y]$, while ElGamal [ElG84] allows multiplications, i.e., $[x \cdot y] = [x] \odot [y] = [x] \cdot [y]$. Unfortunately, these schemes do not allow for the respective other arithmetic operation. This limits their use in secure protocols to either simple applications or requires interaction for more complex computations. Nevertheless, PHE has been successfully applied, e.g., to face recognition [EFG+09], recommender systems [EVTL12], or signal processing [LEB13].

**Somewhat Homomorphic Encryption (SWHE)** schemes [BGN05, BGV12] allow both addition and multiplication in the encrypted domain. However, homomorphic operations generate noise that accumulates such that the result eventually cannot be correctly decrypted anymore, which limits the number of subsequent executions of one of the two homomorphic operations. These schemes are thus not fully homomorphic, but often referred to as leveled.
Fully Homomorphic Encryption (FHE) schemes, named “cryptography’s Holy Grail” [Mic10], were discovered only in 2009 [Gen09, VDHV10, BV14]. FHE allows an unlimited number of additions and multiplications in the encrypted domain. Being functionally complete, FHE can theoretically be used to evaluate any computable functionality securely by evaluating the corresponding arithmetic circuit of additions and multiplications. The primary application of FHE is in outsourcing computations over sensitive data to an untrusted cloud, e.g., as demonstrated for genetic testing [ZDJ+15, KL15b, LYS15]. Since the applicability of FHE schemes is limited by their significant processing and storage overheads, PHE and SWHE schemes still have a raison d’être.

Overview: We now explain how to build generic secure protocols on top of HE, concentrating on the two-party case which is the main application scenario of such schemes in this thesis. As in the LSS-based protocols, the desired functionality \( F(\cdot) \) is first expressed as an arithmetic circuit \( F_{\text{arith}}(\cdot) \) of addition and multiplication gates which is subsequently evaluated by the two parties (we call them client \( C \) and server \( S \) to emphasize the asymmetry of the protocol) in three phases.

1) Encrypting the inputs: \( C \) encrypts all its inputs and uploads them to \( S \).

2) Evaluating the arithmetic circuit: \( S \) evaluates the arithmetic circuit under encryption using \( C \)'s encrypted inputs and its own unencrypted inputs, interacting with \( C \) only when necessary.

3) Decrypting the results: Depending on who should learn the final result, \( C \) decrypts and learns the result or blindly decrypts it for \( S \) and does not learn it.

We describe each phase in more detail and summarize the state of the art afterwards.

Encrypting the inputs: At any time before the computation, \( C \) generates a key pair for the chosen HE scheme and encrypts all its inputs with the private key. When the computation starts, \( C \) sends the public key and the encrypted inputs to \( S \).

Evaluating the arithmetic circuit: Since the combination of addition and multiplication is functionally complete, it suffices to show how \( S \) can evaluate these operations under encryption.

Addition: Addition can be computed locally by \( S \) given an additive HE or SWHE or any FHE scheme. (Secure protocols over multiplicative homomorphic schemes such as ElGamal [ElG84] are less common, not used in this thesis, and are, thus, omitted.)

Multiplication: Given an FHE scheme multiplications can also be executed locally by \( S \). For a limited amount of sequential multiplications, this also holds for SWHE schemes. However, when an additive homomorphic FHE scheme is employed, e.g., for efficiency reasons, \( S \) requires \( C \)'s help to carry out encrypted multiplications [KSS10]. Given encryptions \([x]\) and \([y]\) under \( C \)'s public key, the server first blindly them additively with two random numbers \( r_1, r_2 \in \mathcal{P} \) from the plaintext domain, i.e., obtains \([x'] = [x + r_1] = [x] \oplus [r_1] \) and \([y'] = [y + r_2] \) likewise. Note that \( x \) and \( y \) are now perfectly blinded over the plaintext domain and can be sent to \( C \) without \( C \) learning anything about these values. This is necessary to hide not only the inputs but also any intermediate values.
of the computation from $C$, according to SMC security definitions (Section 2.2). $C$ decrypts both blinded values $x'$ and $y'$, multiplies them, encrypts the result, and sends $[x'y']$ back to $S$ who computes

$$[x'y'] \oplus [r_1r_2] = [xy + r_2x + r_1y + r_1r_2 - r_2x - r_1y - r_1r_2]$$

$S$ can locally compute the terms $[-r_2x] = [x]$ and $[-r_1y] = [y]$ since it knows $r_1$ and $r_2$ in clear. Negation of $r_1$ and $r_2$ is performed within the group that defines the plaintext space of the chosen HE scheme.

**Decrypting the results:** After evaluating the whole arithmetic circuit gate-by-gate, $S$ obtains the encrypted final result $[z]$. If only $C$ should learn the result, $S$ sends $[z]$ and $C$ decrypts. If only $S$ should receive the result, $S$ sends the blinded result $[z + r]$, $C$ decrypts and sends back $z + r$ which $S$ only needs to unblind. If both should learn the result, $C$ decrypts it and additionally provides it to the server afterwards.

**State of the art:** As in the LSS-based protocols, the main overhead in PHE-based secure protocols stems from the interactive multiplication steps and conceptually similar optimizations apply, i.e., precomputed multiplication triples [DPSZ12, DKL+13], parallelism [DGKN09, HSS+10], and size efficient building blocks [CDH10, CS10]. A second major source of overheads are the big integer operations that are inevitable in HE. For example, the popular Paillier scheme [Pai99] requires key lengths of 1024 bit to 3072 bit for short to long-term security [BBB+07] and the resulting ciphertext space has twice this bitlength. In contrast, security of secret sharing is independent of the underlying field such that we can use small size polynomial fields of, e.g., 32 bit or 64 bit, whose elements can be processed directly within the processor’s registers.

As the plaintext space in all HE flavors are usually much larger than the actually occurring input values, an important optimization is to pack multiple values into one ciphertext and then to simultaneously operate on all packed values by just modifying the single ciphertext. For example, we can pack 64 standard 32 bit integers into one ciphertext when using medium-sized 2048 bit keys in Paillier. This has opened up a line of research that targets the optimization of secure protocols for Single-Instruction-Multiple-Data (SIMD) type of operations [BPB10, BFL+11a, EVTL12] and has brought significant improvements especially to FHE schemes [SV10, SV14, GHS12]. Packing techniques, however, require precise knowledge of input sizes and the applied arithmetic operations in order to avoid numeric overflows among the packed values that would render computed results completely useless. Finally, multiple approaches have been proposed to achieve security against malicious adversaries at the price of significant additional overheads [BDOZ11, DPSZ12, DKL+13, KSS13a, ABB16].

### 2.3.4 Notation

Part of the contributions presented in this thesis are novel or improved protocol designs based on the approaches discussed so far. In this section, we thus summarize the most important notation for later reference.
2.3. Generic Secure Protocols

Protocol parties: All protocols are carried out between parties $P_1, ..., P_m$ with $m \geq 2$. We only deviate from this notation and assign special names when parties have designated or special roles, e.g., sender and receiver in OT.

Messages and communication: A single message sent from $P_i$ to $P_j$ is written as $P_i \Rightarrow P_j$, while $P_i \leftrightarrow P_j$ denotes multiple messages and rounds of communication between the two parties.

Oblivious Transfer: $(s_1, x_1, ..., s_m, x_m) \leftarrow 1:n$-OT denotes an Oblivious Transfer between a sender $S$ and a receiver $R$, where $R$ obtains $m$ secrets $s_i, x_i$ of his choices $r_i \in \{1, ..., n\}$ through a batch of $m$ parallel OTs with each OT carried out over $n$ secret strings of bit length $l$. When embedding OT as a building block within another secure protocol, we only explicitly note which party acts as sender $S$ and receiver $R$ when it is not clear from the context.

Garbled Circuits: $(y_1, ..., y_m) \leftarrow GC_{F_{bool}}(x_1, ..., x_n)$ denotes the secure evaluation of a Boolean circuit $F_{bool}(\cdot)$ on inputs $x_1, ..., x_n$ using Yao’s GC protocol with outputs $y_1, ..., y_m$ between the garbler $G$ and the evaluator $E$. This involves all steps of Yao’s protocol, i.e., garbling the circuit and inputs, evaluating the circuit, and ungarbling the results. All inputs and outputs can be plaintexts $x$, garbled values $\tilde{x}$, or additive shares $\langle x \rangle$. For the sake of simplicity, we do not explicitly note who supplies which input and obtains which output when this is clear from the context.

GMW: We write $(y_1, ..., y_m) \leftarrow GMW_{F_{bool}}(x_1, ..., x_n)$ for the evaluation of a Boolean circuit $F_{bool}(\cdot)$ on inputs $x_1, ..., x_n$ using the GMW protocol with outputs $y_1, ..., y_m$ between two parties, $P_1$ and $P_2$. Inputs and outputs can be plaintexts $x$ or Boolean shares $\langle x \rangle$. An individual share owned by party $P_i$ is denoted by $\langle x \rangle_i$. For the sake of simplicity, we do not explicitly note who supplies which input and obtains which output when this is clear from the context.

Secret Sharing: Continuing this notation, $(y_1, y_2, ..., y_m) \leftarrow LSS_{F_{arith}}(x_1, ..., x_n)$ denotes the evaluation of an arithmetic circuit $F_{arith}(\cdot)$ on inputs $x_1, ..., x_n$ with outputs $y_1, ..., y_m$ between $m \geq 2$ parties, $P_1, ..., P_m$. Inputs and outputs can be plaintexts $x$ or arithmetic shares $\langle x \rangle$, with $\langle x \rangle$ the shared input and $y_i$ the individual output of party $P_i$. For the sake of simplicity, we only note explicitly that a sharing is Boolean or arithmetic when it is not implied by the context. Since arithmetic circuits are intuitive to read, we mostly write them directly using operators $\oplus$ and $\odot$ for addition and multiplication in the respective LSS scheme. For example, the notation $(y) = LSS_{arith}(\langle a \rangle, \langle b \rangle, \langle c \rangle, \langle d \rangle)$ translates to the more intuitive expression $(y) = \langle a \rangle \oplus \langle b \rangle \odot \langle c \rangle \odot \langle d \rangle$.

Homomorphic Encryption: Finally, we write $(y_1, ..., y_m) \leftarrow HE_{F_{arith}}(x_1, ..., x_n)$ for the evaluation of an arithmetic circuit $F_{arith}(\cdot)$ between the client $C$ and the server $S$. Inputs and outputs can be plaintexts $x$ or homomorphically encrypted ciphertexts $\tilde{x}$. As before, we write arithmetic expressions directly using operators $\oplus$ for addition and $\odot$ for multiplication in the respective HE scheme. For example, $[a] \oplus [b] \odot ([c] \odot [d])$ means that we compute $a + b \cdot (c + d)$ on ciphertexts using the techniques for encrypted addition and multiplication described above.
Secure Multiparty Computation

Security Level: The security level denotes the number of operations that are required to break the security of a cryptographic primitive or protocol, e.g., through a brute-force attack. A security level of \( t \) bits means that any attack is expected to take \( 2^t \) operations on average. In this thesis, we commonly refer to short-term, medium-term, and long-term security for 80-bit, 112-bit, and 128-bit security levels as summarized in Table 2.1.

For symmetric cryptography and hashing, \( t \) usually directly corresponds to the bitlength of the secret keys or hashes. We refer to the National Institute of Standards and Technology (NIST) recommendations [BBB+07] to establish key lengths for asymmetric security primitives of equivalent security, e.g., \( T = 1024 \) bit, \( 2048 \) bit, and \( 3072 \) bit keys for asymmetric primitives based on the hardness of integer factorization (e.g., RSA), and key lengths of size \( T' = 160 \) bit, \( 224 \) bit, and \( 256 \) bit when security is based on the hardness of the discrete logarithm, e.g., in elliptic curve cryptography.

Note that finite security levels require computationally-bounded adversaries (cf. Section 2.2). Hence, security against unbounded adversaries can be intuitively denoted by a security level of \( t = \infty \).

### Table 2.1 Summary of security levels used in this thesis and the related literature: Numbers denote minimum required key lengths in bits according to NIST recommendations [BBB+07].

<table>
<thead>
<tr>
<th>Security Level</th>
<th>Symmetric Cryptography</th>
<th>Factoring Modulus</th>
<th>Discrete Logarithm</th>
<th>Hash Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>short term</td>
<td>80</td>
<td>1024</td>
<td>160</td>
<td>160</td>
</tr>
<tr>
<td>medium term</td>
<td>112</td>
<td>2048</td>
<td>224</td>
<td>224</td>
</tr>
<tr>
<td>long term</td>
<td>128</td>
<td>3072</td>
<td>256</td>
<td>256</td>
</tr>
</tbody>
</table>

2.4 Instantiations and Applications

In the previous section, we have taken a rather theoretical view on the feasibility of secure computations focusing on the construction of generic secure protocols based on different cryptographic primitives and their most important optimizations. In this section, we take a much more practical view and shift our focus to actual instantiations and implementations of these techniques that target the applicability of secure computations at increasing levels of abstraction. To this end, we briefly survey different secure computations frameworks, domain-specific languages and compilers, as well as proposed applications.

#### 2.4.1 Frameworks and Libraries

The lowest level of abstraction is achieved by secure computation frameworks and libraries that allow developers to compose secure protocols based on different provided higher level building blocks such as arithmetic and logical operations, multiplexers, or vector operations. Early proposals include two party frameworks based
on GC [MNPS04, HEKM11] or the GMW protocol [CHK+12, DSZ14], multi-party frameworks based on LSS [BSMD10, BLW08, ABZS13, ABB16], as well as different PHE-based libraries [Tod13, EVTL12].

These first experiences have lead to two major insights. First, they showed that GCs and GMW, being deeply rooted in Boolean logic, are very efficient for logical operations, while the HE and LSS approaches, defined over arithmetic circuits, proved much more efficient for arithmetic operations. In conclusion, mixed protocols were proposed that combine secure computation techniques over Boolean and arithmetic circuits. The basic idea of these approaches is to build efficient conversion protocols and execute required operations in the most efficient representation. HE and GCs were first combined in [BPSW07] followed by different other frameworks [HSS+10, SK11, KSS13b, KSS14], while the most recent ABY STC framework [DSZ15] combines GCs, GMW, and additive secret sharing. The presented results show that mixed protocols can significantly improve the efficiency of secure computations.

Second, the early frameworks showed that accessing memory at private locations, e.g., selecting element $A[i]$ from an array $A$ where the data as well as the access location $i$ are private, is very expensive in all approaches. This has recently lead to the adoption of oblivious data structures to secure computation frameworks, e.g., the framework proposed in [LWN+15] builds upon Oblivious RAM [Gol87, GO96, WCS15]. Through oblivious data structures, the performance of secure computations was improved by one order of magnitude in data intensive applications [LWN+15].

Most recently, various FHE libraries have been developed, e.g., HElib [HS13, HE16], SEAL [LCP16], and FHEW [DM15, FHE14] that aim to make FHE schemes practical.

### 2.4.2 Languages and Compilers

Frameworks and libraries, allowing for quick extensions and optimizations, are popular among researchers and developers with a strong background in cryptography and knowledge of secure computation techniques. In contrast, a standard application developer with little or no prior experience in secure computations is likely to be overwhelmed by them [PGFW14]. The reason is that the building blocks provided in proposed frameworks and libraries are still rather low-level. For example, application developers can choose between many powerful statistics and machine learning libraries (without security), while one of the best developed STC frameworks, ABY [DSZ15], not even supports arithmetic over rational or real data types; piecing together fixed or floating point primitives, e.g., from the approaches proposed in [CS10, ABZS13], and building desired statistics or machine learning algorithms on top is too much to ask of a developer who regards secure computations as a tool and not as a research topic.

This insight has lead researchers to target a higher level of abstraction, e.g., special compilers that transform programs in standard or domain-specific programming languages into secure protocols. Interestingly, the need to define higher level abstractions to support inexperienced developers was already recognized by the seminal Fairplay framework [MNPS04], which allowed to specify programs in a (very constrained) domain-specific language, the Secure Function Definition Language [MNPS08] that was later optimized in [Ker11, MLB12].
The idea of defining an own domain-specific language has been adopted in other proposals. The Picco compiler \cite{ZSB13} compiles an extension of C to a LSS-based multi-party computation protocol. Obliv-C \cite{ZE15} is a C extension with support for oblivious data structures that can be compiled using a GCC wrapper. The language and compiler by Kreuter et al. \cite{KSS12} provides security in the malicious model.

Other proposals focus on compiling secure protocols directly from standard programming languages (mostly C). CBMC-GC \cite{HFKV12,BKJK16} compiles C code into Boolean circuits and uses the CBMC model checker to automatically minimize the circuit size. Similarly, the Portable Circuit Format \cite{KSMB13} is also based on Boolean circuits and can be compiled from C. TinyGarble \cite{SHS+15} and Demmler et al.’s approach \cite{DDK+15} compile Boolean circuits from Verilog or VHDL in order to take advantage of (commercial) logic synthesis tools for circuit minimization.

### 2.4.3 Benchmarks and Applications

The presented frameworks, libraries, and compilers have lead to the realization of different widely used benchmarks to compare their performance. One of the most used benchmarks is privacy preserving AES, where one party holds the encryption key \( k \), the other party holds a block \( B \), and both want to compute \( \text{AES}_k(B) \), the AES encryption of the block \( B \) under key \( k \) \cite{PSSW09,HSS+10,HEKM11,KSS12,SHS+15}. Another popular benchmark is private distance computation between two inputs held by two distinct parties, e.g., hamming, edit, or euclidean distance, \cite{HSS+10,KSS12,KSMB13,ZSB13,DSZ15,SHS+15,LWN+15}. Further benchmarks include standard arithmetic operations over integers and fixed or floating point numbers \cite{KSMB13,ZSB13,SHS+15}, matrix multiplication \cite{KSMB13,ZSB13,SHS+15}, set intersection \cite{DSZ15}, or super linear algorithms such as sorting \cite{ZSB13} and Dijkstra’s shortest paths \cite{LWN+15}.

These benchmark functionalities have been used as building blocks for more complex applications that combine multiple of these techniques and extend upon them. We highlight several of these applications sorted by their respective application field, i.e., i) biometrics, ii) pattern recognition, machine learning, and signal processing, iii) finance, and iv) e-health.

In biometrics, a typical task is to recognize a human by biometric features. Scenarios emerge where one party holds the database of biometric feature against which the other party wants to authenticate in a privacy preserving manner. Particular proposals include privacy preserving variants of fingerprint matching \cite{EHKM11,ZSB13}, face recognition \cite{EFG+09,SSW09,OPJM10}, or speaker verification \cite{PR13}. A survey of the application of secure protocols in biometrics is given in \cite{BCP13}.

In privacy preserving pattern matching, machine learning, and signal processing usually one party holds a stochastic model of some sorts and the other holds the input that should be matched, classified, or otherwise processed by the other party’s model. Examples include speech processing \cite{PR13,LEB13}, computations over Hidden Markov Models \cite{PRSR11,FDH+11,AB13}, ridge regression \cite{NWI+13}, and machine learning classification \cite{BFL+11a,BPTG15,DGBL+16}.
In finance, applications of secure protocols are electronic auctions [BCD+09], the analysis of confidential financial data [BTW12], detecting tax fraud [BJSV15], or privacy preserving credit ranking [DDN+15].

Applications in e-health, medical research, and healthcare have focused on genome-wide association studies [KBLV13,KL15b,CTW+15,ZBA15,KBLV13,ZDJ+15]. Further examples include statistical genomic tests [DCFGT12,ARM+13,DDC14] and the privacy preserving search for patients with similar traits and symptoms [WHZ+15,KL15b,ZBA15].

Other notable applications of secure protocols are privacy preserving prediction of satellite collisions [KW15], route mapping with secret maps, sources, and destinations [CLT14], and encrypted Voice-over-IP streaming with audio compression [AR15].

2.5 Summary

In this section, we first motivated SMC from a classical distributed systems’ point of view (Section 2.1) and then presented a formal model together with rigorous security definitions (Section 2.2). In this model, the goal of secure protocols is to emulate an ideal world, where a perfectly trusted and incorruptible third party helps in the computation, through cryptography in the real world, where no such third party is available. We then explained four constructions of generic secure protocols that achieve security in this very restrictive model (Section 2.3). All four approaches first reduce the desired functionality to either a Boolean circuit (GC and GMW) or an arithmetic circuit (LSS and HE), use some form of cryptographic protection for sensitive inputs, and show how to evaluate a functionally complete set of operators on such protected values. While these constructions were only of theoretical interest first, two decades of optimizations and general advances in computing and communication technology have enabled several practical applications (Section 2.4).

The four presented constructions, in particular efficient combinations of them, build the technical foundation for our own contributions made in this thesis – they are used as building blocks and their presentation has been kept high level for the sake of brevity. Clearly, the discussed techniques also contribute towards the overall goal of this thesis, the technical maturity and adoption of SMC and STC, and thus address one or more of our research questions that we have raised in the introduction regarding the functionality, efficiency, and customizability of secure protocols. We thus provide a detailed analysis of related work in the Chapters 3, 4, and 5 in order to clearly mark out the contributions made in this theses. This being said, we now briefly review our own research questions in the light of the discussed technical background including the most recent developments to conclude this chapter.

Motivating Problems and Use Cases: A multitude of applications have been built on top of generic secure protocols in recent years. Many of them focus only on privacy protection and fail to incentivize service providers. Thus, only few proposals have resulted in real-world deployments or commercial products [ABPP16], e.g., key management [Dya17,PAR], auctions [PAR], and analysis of financial data [Cyb]. This shows that the need for compelling use cases
that are technically feasible and, ideally, non-zero sum still persists. To address this need, we present three novel applications of SMC, i.e., mixing of digital currencies (C1), indoor localization (C2), and crowd-sensing (C3). We, furthermore, continue and improve upon previously recognized applications, i.e., pattern recognition and machine learning (C2) as well as genetic testing (C3).

Q1 Functionality: The development of domain-specific languages and compilers (Section 2.4.2) has progressed in the last years such that simple algorithms and functionalities can today be implemented almost without expert knowledge. However, SMC functionality is nowhere near to being equivalent to the functional spectrum offered by standard programming languages and libraries. The multitude of special purpose protocols that have been proposed shows that more complex functionality is still infeasible when realized in generic frameworks. In our oblivious shuffle scheme and its verification mechanism (C1), different building blocks for computing over probabilities (C2), and our location privacy algorithm (C3), we balance generic approaches with special purpose optimizations to realize two examples of such novel functionality.

Q2 Efficiency: All generic constructions for SMC in their original form suffered from prohibitive performance overheads. Significant improvements have been made through both conceptual optimizations as well as practical engineering. Still, processing and communication overheads limit which SMC applications are practical. Thus, performance remains one of the single most important optimization goals and shows the great importance of our third research questions. We contribute efficient secure protocols for pattern recognition and machine learning (C2) and outsourced set intersection (C3) that balance generic and special purpose constructions to outperform the state of the art.

Q3 Customization: All of the constructions of generic secure protocols, that we have explained earlier, assume fixed setups of the protocol parties in stable networks and most state of the art optimizations continue with this setting. Only few proposals optimize for different settings, e.g., involving mobile devices [DSZ14], or propose customizable trade-offs, e.g., between processing and communication [DKS17]. In order to customize SMC to constrained deployment and operation scenarios, we propose bandwidth-optimized building blocks (C2) as well as a secure outsourcing (C2, C3).

In view of the broad theoretical foundations and the research efforts before, during, and after our own, the core of all our contributions is to build complete secure computation systems that integrate, improve, or extend upon state of the art techniques. Many of the used techniques have been proposed individually and been claimed compatible, scalable, accurate, and secure. However, it is only when we incorporate them in a complete system and validate their applicability in real-world use case that these attributes can be fully proven. Perhaps surprisingly, this is a major effort in many cases – the devil is in the details.
Decentralized Mixing of Digital Currencies

In this chapter, we consider decentralized digital currencies as a promising application area for Secure Multiparty Computation (SMC) to provide rigorous security and financial privacy. Following our use case-driven research methodology, we first motivate the problem field of financial privacy in the context of secure computations and identify a compelling use case in the secure mixing of decentralized digital currencies such as Bitcoin (Section 3.1). In our problem statement, we distill core requirements for ideal mixing services, present a detailed analysis of prior works, and summarize our notable contributions (Section 3.2). As the main part of this chapter, we present CoinParty [ZGH\textsuperscript{+}15, ZMH\textsuperscript{+}16, Gro14, Mat15], a novel decentralized mixing system secured through SMC that addresses the identified problems and improves significantly on prior works (Section 3.3). We qualitatively and quantitatively evaluate our prototype implementation and validate its real-world applicability in a cloud setup (Section 3.4). We conclude this chapter with a short discussion of recent developments and possible future work (Section 3.5).

3.1 Motivation

Digital payment schemes have been widely adopted, due to the rise of e-commerce and the proliferation of smart phones among other reasons. They allow convenient remote payments and are cheaper to provide and maintain than cash money. However, they also have a significant downside in terms of privacy. Prevalent digital payment schemes are realized through centralized services (i.e., banks or bank-like entities), which are single points of failure and thus an inherent security risk. Furthermore, each issued transaction is visible to and logged by the service providers who are thus not only in a position to allow or block certain transactions but are also able to gain (and monetize) deep insights into their users’ personal lives.
Conventional cash money, in contrast, provides individuals with the freedom to engage in monetary transactions without knowledge or permission of any third party. It is currently an open discourse whether these are values that should be upheld or abolished. On the one side, anonymous payments are often connected with illicit work, tax fraud, black markets, and with criminal activities in general \cite{MPJ+13}. On the other side, anonymous payments are recognized as an important means for citizens to exercise their right for informational self-determination and are as such an integral part of a free and democratic society as argued by the scientific advisory council of the German Federal Ministry for Economic Affairs and Energy \cite{Ger17}. Since digital payment schemes have already banished anonymous cash payments from many areas of our everyday lives and the continuing digitalization further fuels this process, the question is not whether we need to uphold cash money but rather whether we can design a digital equivalent of it, at all?

An increasingly popular approach to this challenge are decentralized cryptographic currencies \cite{BMC+15,TS16}. Cryptocurrencies are often motivated from a distrust in central banks, which explains their most characteristic and name-giving feature, the decentralization of all control: In decentralized cryptocurrencies, as opposed to traditional currencies, any control is completely removed from centralized entities and is, instead, placed into the collective hands of a distributed network or community of peers between whom no trust exists. In particular, in decentralized digital currencies no single centralized entity can control the creation or the flow of currency.

The core motivation, assumptions, and trust models of cryptocurrencies are very similar to those of SMC: Centralized banks correspond to the incorruptible Trusted Third Party (TTP) assumed in the imaginary ideal world of the SMC model (cf. Section 2.2) – except that banks are far from being incorruptible in practice, not least due to national economic and safety policy. In contrast, cryptocurrencies are similar to SMC protocols in their attempt to emulate the ideal world through distributed cryptographic protocols in the real world where no incorruptible TTP exists.

Besides these similarities, we also observe a crucial difference between the proposed cryptocurrencies and SMC protocols: The development of cryptocurrencies has, so far, not followed the rigorous model and security definitions of SMC and their promises of correctness, robustness, and privacy have not been formally proven. The incorporated privacy measures of cryptocurrencies have, thus, been subject to different attacks that successfully de-anonymize users \cite{AKR+13,RH11,MPJ+13,BP15}. A prominent approach to reestablish privacy in cryptocurrencies is to mix digital cash of a sufficiently large group of users to obfuscate money flows from any adversarial observer. Unfortunately, such mixing services have been realized as central entities which poses a security and privacy threat, e.g., the service could be compromised by hackers \cite{Her14}, subpoenaed by governments \cite{Lad14}, or simply scam users’ out of their Bitcoins \cite{Bit16}.

Our core idea and novel use case of SMC is to emulate centralized mixes of digital currencies through distributed cryptographic protocols without the need for a TTP. In this, we follow in the spirit of cryptocurrencies that any control should be decentralized and any (centralized) accumulation of sensitive data should be avoided. By developing our system within the strict and well seasoned security model of SMC, we can rigorously secure the involved monetary transactions as well as provide strong guarantees of privacy against any insiders and outsiders.
3.2 Problem Analysis

In this section, we analyze the problem of mixing digital currencies in detail by the example of the decentralized cryptocurrency Bitcoin, which was proposed in 2008 as an anonymous alternative to centralized digital payment schemes [Nak08]. Today, Bitcoin is the most widely adopted decentralized cryptocurrency (cf. Figure 3.1) with a staggering market capitalization of $43.00 Billion in July 2017 [Bit17a] and many other proposed cryptocurrencies have been derived from its design. Bitcoin is thus a reasonable choice to motivate and demonstrate our application of SMC to mixing digital currencies.

We first provide a short overview of the most important features of Bitcoin required to understand the remainder of this chapter (Section 3.2.1). We point out threats to Bitcoin’s privacy and identify mixing services as promising yet immature approaches to thwart these threats (Section 3.2.2). Our problem statement distills ideal security requirements and design goals for mixing services (Section 3.2.3) against which we pit existing approaches (Section 3.2.4) as well as our own contributions (Section 3.2.5).

3.2.1 A Bitcoin Primer

Bitcoin is best understood as a decentralized peer-to-peer network that keeps track of all money transfers between its users. This is achieved through three core components, i) the blockchain, ii) account addresses, and iii) monetary transactions.

Blockchain. To keep track of the balances and to establish trust in the currency, all Bitcoin transactions are stored in a distributed public append-only ledger, the blockchain. The blockchain thus contains the complete history of accepted Bitcoin transactions and is maintained and verified collaboratively by the Bitcoin network. Adding a new block to the blockchain requires a proof-of-work such that diverging from the blockchain at an earlier point in time would require redoing all proof-of-works for the successor blocks. This protects against tampering and rules out double-spendings of bitcoins as long as the majority of computation power is contributed by honest participants. The blockchain thus serves as a mechanism to reach a decentralized consensus about the state of the Bitcoin system, in particular the ownership of bitcoins.
3. Decentralized Mixing of Digital Currencies

Addresses. To receive, store, and spend bitcoins, users maintain Bitcoin addresses which are, in the simplest case, the hash of a public key for the Elliptic Curve Digital Signature Algorithm (ECDSA) [Nat94]. Each address can be thought of as a very simple bank account which belongs to and can be accessed by anyone who knows the corresponding private key. Bitcoin uses the secp256k1 elliptic curve [SEC10] which provides approximately $2^{256}$ possible keys. Users can thus generate and maintain virtually unlimited amounts of addresses for themselves.

Transactions. A transaction is a command to transfer bitcoins from one address, the input address, to another address, the output address. To prove that she is authorized to spend the funds stored at the specified address, the issuer digitally signs the transaction using the ECDSA scheme [Nat94] and the private key corresponding to the input address. The signed transaction is broadcasted to the Bitcoin network, verified by the other peers, and is eventually included in the blockchain as part of a fresh block. A new transaction is accepted by other Bitcoin users when six subsequent blocks have been processed and accepted after which a diversion of the blockchain is considered to be practically ruled out [Ros14].

Beside the simplest form of one-to-one transactions, Bitcoin also allows group transactions, i.e., transactions from multiple input addresses to multiple output addresses, which may be owned by different users. A group transaction is only accepted by the Bitcoin network if it has been correctly signed with all private keys corresponding to the specified input addresses. This means that a group transaction can only ever be executed in its entirety or not at all, but never in parts. Just as standard one-to-one transactions, group transactions are thus atomic operations in the Bitcoin network.

Importantly, a transaction always redeems all funds available at the inputs. If less than the full amount is transferred to the output addresses, the difference can be collected as a transaction fee. Transaction fees incentivize miners, i.e., peers who invest computing power to carry out the proof-of-work required for the creation of new blocks, to include the transaction in the blockchain. Historically, transaction fees were voluntary, while non-negligible fees proportional to the transaction size have become necessary due to Bitcoin’s recent rise in popularity [Bit17c].

3.2.2 Financial Privacy in Bitcoin

To participate in the Bitcoin network, users are neither required to register and set up an account nor to maintain any kind of stable identity. All balances are kept solely on the basis of addresses which can be generated locally and on-demand by any user. To transfer money, a user does not need to know the recipient’s identity but only the destination address. In consequence, payer and payee are anonymous as long as the input and output addresses of a transaction cannot be linked to them. Addresses thus serve as pseudonymous cryptographic identifiers and are the primary mechanism for anonymity built into Bitcoin. Users are officially encouraged to use fresh unlinkable addresses as often as possible to ensure their own anonymity [Nak08].

Especially the promise of financial privacy has drawn great interest towards Bitcoin and related cryptocurrencies. Different studies [AKR+13, RH11, RS13, MPJ+13], however, raise serious doubts about how much anonymity Bitcoin’s pseudonymous addresses really provide. These studies show how transactions and addresses of
3.2 Problem Analysis

Individual users can often be linked together using taint analysis and clustering techniques on the transaction graph in the publicly available blockchain. Even worse, the techniques presented in [KKM14, BP15] show that addresses can be linked to IP addresses, thereby completely de-anonymizing their owners.

These works raise the question how Bitcoin users can reestablish their financial privacy. Of course, users can generate fresh unlinkable addresses at any time. Transferring funds from a de-anonymized address to such a fresh address, however, links both addresses together in the blockchain and exposes the owner again to the different de-anonymization methods demonstrated in [AKR13, RH11, RS13, MPJ13]. Therefore, a mechanism is required to send funds from a de-anonymized address to a fresh address in an unlinkable manner.

The most prominent approach to solving this problem are mixing services as illustrated in Figure 3.2: The four users, whose Bitcoin addresses $I_A, ..., I_D$ have been linked to their person, escrow their funds to a mixing service. They also each generate a fresh unlinkable output address $O_A, ..., O_D$, and confidentially (e.g., by encrypting output addresses with the mixing service’s public key) tell the mixing service to return mixed funds to these addresses. The mixing service pools the funds of sufficiently many users, shuffles them secretly, and repays each user on her individual output address after a random waiting period in order to thwart attackers that try to correlate users’ input and output addresses in time. Through the mixing, the de-anonymized input address of each individual user remains unlinkable to her fresh output address to any observer of the blockchain without the knowledge how addresses were shuffled by the mixing service.

Conceptually, the idea of mixing bitcoins of different users effectively counters the de-anonymization approaches described in [AKR13, RH11, RS13, MPJ13]. Practical deployments of mixing services are proving challenging, though: Centralized mixing systems are straightforward to realize, but require the user to fully trust the service operator not to steal the mixed funds [Bit16, Bit17b] and not to de-anonymize users [Lad14, The14]. Decentralized mixing systems [Max13a, Yan12, RMSK14] provide better security guarantees but face scalability and usability issues and, due to characteristic patterns in their mixing transactions, achieve less anonymity.
3. Decentralized Mixing of Digital Currencies

3.2.3 Problem Statement

Motivated by the lack of privacy in Bitcoin demonstrated by the variety of de-anonymization attacks and the observed shortcomings of existing mixing services, we re-consider the problem how users can securely and anonymously mix their funds to reestablish and preserve their financial privacy.

We abstractly define the problem of mixing as follows: We consider $n$ users who each dispose of at least $\nu$ bitcoins ($\mathcal{B}$) stored at their input addresses $I_1, \ldots, I_n$. The users want to randomly mix the fixed amount of $\nu \mathcal{B}$ to a set of output addresses $O_1, \ldots, O_n$ such that i) each user $U_i$ receives back $\nu \mathcal{B}$ on her output address $O_i$, and ii) output addresses are unlinkable, i.e., only user $U_i$ knows that $O_i$ belongs to her.

We abstractly refer to a system that solves this problem as a mixing service and the single or multiple parties that provide the service as mixing peers. An ideal mixing service should fulfill the following requirements and design goals:

**Correctness:** Bitcoins must not be lost, stolen, or double-spent by any inside or outside party even in the presence of a malicious adversary. As the result of the mixing, each user should receive the paid funds on her output address (less an arbitrary, optional service fee). The mixing service should eventually succeed to mix funds of honest users, i.e., be resilient to random failures and deliberate protocol-level Denial of Service (DoS) attacks from any inside or outside party. The mixing should be random and no inside or outside party should be able to bias, reduce, or predict the randomness.

**Anonymity:** Mixing funds from input to output addresses must be anonymous, i.e., a malicious adversary must not be able to link any user $U_i$ to her output address $O_i$ by any means other than guessing. The anonymity of a mixing is measured in the size of the anonymity set, which is formed by the group of users among which an adversary has to guess. The size of the anonymity set should be maximized to decrease the attacker’s chances of guessing the correct output address.

**Deniability:** Users should be able to plausibly deny having participated in a mixing operation. This implies that a mixing operation should not produce any irrefutable evidence that binds a user to a mixing operation.

**Scalability:** Since the anonymity of mixings increases with the number of participants, mixing services should scale to large numbers of users and render participation as easy and cheap as possible. Furthermore, the impact on the Bitcoin network, e.g., the number of issued transactions, should be minimized.

**Cost-efficiency:** The protocol should be cost-efficient in terms of the involved mixing and transaction fees. The collection of mixing fees should be optional.

**Applicability & Usability:** The mixing service should be fully backwards-compatible with the current Bitcoin network and allow an incremental deployment. The service should be easily accessible and usable, i.e., user interaction should be minimized and require only standard software such as a Bitcoin client.

In the next section, we discuss existing mixing services and analyze in detail to what extent they address these requirements and design goals.
Table 3.1 Comparison of CoinParty to a representative selection of prior works by the stated requirements and design goals. For anonymity, we provide the size of the anonymity set a user is guaranteed against the mixing service (left), the other participants in the mixing of which might be compromised (middle), and outside observers (right) – anonymity sets should be maximized.

3.2.4 Related Work

We divide our analysis of existing mixing services into two categories, i) those that use a centralized architecture [Bit14a, Bit14b, BNM+14, VR15] and ii) those that use a distributed architecture [Max13a, RMSK14, Ros12, Yan12, MGGR13, SCG+14]. Approaches in one category share key characteristics. The results of our analysis are summarized in Table 3.1 and compared to our own approach, CoinParty. We prove and discuss all properties of CoinParty in detail in Section 3.4.

3.2.4.1 Centralized Mixing

The first generation of commercial mixing services [Bit14a, Bit, Coi] were centralized, i.e., a single entity receives all funds and carries out the mixing as illustrated in Figure 3.2. Central mixes have obvious advantages: they are straightforward to deploy, easy to use, and scale to thousands of users. Indeed, as of April 2017 we count several commercial mixing services in operation [Bit14a, Bit17b]. These commercial mixing services effectively anonymize users against any outsider who tries to de-anonymize users by analyzing the transaction graph in the blockchain. In addition, centralized mixing has the following two notable advantages.

First, centralized mixing uses only standard transactions which cannot be distinguished from non-mixing transactions if fresh addresses are used for each mixing operation, termed mix indistinguishability in [BNM+14]. Mix indistinguishability affords plausible deniability and increases anonymity by orders of magnitude against passive observers (i.e., a number $N_\omega \gg n$ of other users are added to the anonymity sets as shown and quantified in Section 3.4.2.2).

A second desirable property of centralized mixing is its complete resilience against attacks from malicious users who try to cause the mixing to fail by supplying incorrect information. Such protocol-level DoS attacks can be executed at very low costs against most existing distributed mixing protocols, e.g., the deliberate abort or inadvertent failure of any user causes the protocols proposed in [Max13a, RMSK14, Ros12, Yan12] to fail.

However, the centralized designs of the first generation of Bitcoin mixes also causes severe drawbacks which are well-known to the Bitcoin community. First, commercial
mixing services do not provide any guarantees that the escrowed funds are effectively mixed [MBB13] or even paid back [Bit16, Bit17b]. Indeed, many allegations of theft and scams circulate the Bitcoin community [Bit16, Bit17b]. Clearly, this does not satisfy our correctness requirement.

Second, the central mixing service chooses and knows the permutation applied to the output addresses as well as which users participated in the mixing. Thus, first generation mixes fulfill our anonymity requirement only against passive outsiders, but not against the operator of the mix itself. The centralized mix then represents a single point of failure that could be breached, incentivized, or forced to reveal how funds were mixed, e.g., to governmental agencies under subpoena [Lad14, The14]. The threat of theft and de-anonymization have lead to widespread distrust in centralized (commercial) mixing services and motivated improvements.

Mixcoin [BNM+14] also has a centralized architecture but improves on first generation mixes by introducing a mechanism to provably blame corrupted mixes for theft of funds. Accountability, the authors argue, incentivizes mixes to behave honestly so as not to damage their reputation. We see different problems with this approach: First, malicious mixes have clear incentives and means to forge a reputation, e.g., the vulnerability of reputation systems to Sybil attacks has been studied in [HZNR09]. Second, proving that a correct mixing has taken place, i.e., to prevent such Sybil attacks, without harming anonymity and deniability of the mixing participants is non-trivial. Third, reputation systems are only reactive and can only punish theft of funds but cannot outright prevent it, e.g., in the case of a mix that after a couple of bad ratings decides to re-open under a new name.

Blindcoin [VR15] directly builds on Mixcoin, but, in contrast to Mixcoin and the other commercial approaches [Bit14a], achieves anonymity against the mixing service itself through the use of blind signatures and a public append-only log. Unfortunately, Blindcoin’s central mechanism to achieve anonymity against insiders at the same time reduces anonymity against outsiders and breaks the deniability property because users leave cryptographic evidence that irrefutably binds their output addresses to the mixing.

### 3.2.4.2 Distributed Mixing

To overcome the prohibitive shortcomings of centralized mixes, a second generation of mixing services [Max13a, RMSK14, has11, Ros12, Yan12, MGGR13], best characterized as decentralized, aim to provide correctness and anonymity even if the mixing service itself is corrupted. A major challenge in decentralized approaches is to protect against theft as well as inadvertent or deliberate protocol aborts. To this end, CoinJoin [Max13a] first proposed to use group transactions for Bitcoin mixings and [RMSK14, Ros12, Yan12] have built on this idea: all mixing transactions are issued in one single atomic transaction with multiple inputs and outputs.

While the atomic nature of group transactions is an elegant solution to the problem of theft, failures, and aborts, they introduce two severe limitations when used for mixing. First, the characteristic form of such group transactions renders mixing transactions easily identifiable in the blockchain [MO15]. Thus, mixes based on the group transaction pattern break deniability and do not achieve mix indistinguishability, i.e., the anonymity sets are limited to the $n$ participants in the atomic mixing
3.2. Problem Analysis

operation. Second, a single malicious user can launch a protocol-level DoS attack on the whole mixing operation simply by refusing to sign the group transaction.

Besides these general disadvantages of using group transactions for mixing, CoinJoin does not offer a solution on how addresses can be shuffled without a central mixing service. In this point, CoinShuffle [RMSK14] improves over CoinJoin and most of the centralized approaches by introducing a distributed oblivious shuffle protocol based on Chaum’s mixnets [Cha81]. As long as at least two of the mixing peers executing the shuffle are honest, users remain anonymous even in the presence of an active attacker compromising other mixing peers. However, even the random failure of a single mixing peer (let alone a deliberate DoS attack) causes the shuffle protocol to fail. Others proposed to use SMC sorting protocols for shuffling [has11,Ros12,Yan12] which could potentially achieve higher resilience against failures and DoS attacks.

Unfortunately, the latter approaches are not expected to scale as SMC sorting is expensive [HEK12,BA16] and they were never thoroughly specified, implemented, and evaluated to prove otherwise.

Further limitations and issues of proposed decentralized mixing services are scalability [Max13b,Yan12,MGGR13,BOLL14], high costs in terms of mixing and transactions fees [Max13b,BOLL14,VR15], or immature protocols and designs [Max13b,has11,Ros12,Yan12]. A notable advantage of all approaches based on the group transaction pattern is that they do not require any escrow of the mixed funds. In contrast to all centralized approaches, a mixing operation is executed in one single big transaction, i.e., at the amortized costs of a single small standard transaction per user. In summary, the second generation of decentralized approaches guarantees anonymity against the mixing service itself but generally achieve much lower anonymity sets, no deniability, and are vulnerable to DoS attacks from insiders — their properties are almost completely polar to those of the first generation mixes.

Departing from all previous approaches, ZeroCoin [MGGR13] introduced a completely different breed of distributed mixes. ZeroCoin extends the Bitcoin system with zerocoins, a second currency with corresponding transactions that is tightly bound to bitcoins: Before spending any zerocoins, a user has to first mint zerocoins from her own bitcoins and supply them to the system. Transactions of zerocoins are unlinkable since they reveal neither the sender nor the receiver and not even the amount of the transaction. This is achieved by proving the correctness of a transaction using Zero-Knowledge Proofs (ZKPs) which do not reveal the particular zerocoin that was spent in the transaction. While the initial approach [MGGR13] does not scale, it was later significantly improved by ZeroCash [SCG\textsuperscript{+}14] and PinocchioCoin [DFKP13]. However, all approaches in this line of research run only as extensions to Bitcoin that would require substantial modifications and at least the majority of Bitcoin nodes to accept those. Since such hard forks are almost impossible in established cryptocurrencies, ZeroCash was launched in 2016 as a full-fledged digital currency completely independent of Bitcoin.\footnote{Zcash. \url{https://z.cash/} (accessed: 2017-07-12)}

3.2.5 Our Contributions

We present CoinParty [ZGH\textsuperscript{+}15,ZMH\textsuperscript{+}16,Gro14,Mat15], a usable, scalable, and fully decentralized Bitcoin mixing service with strong guarantees of correctness,
3. Decentralized Mixing of Digital Currencies

anonymity, and deniability (cf. Table 3.1). Our core idea is to recognize mixing as an SMC problem in the malicious adversary model and hence to investigate the applicability of corresponding SMC techniques. Following the general SMC approach, our goal is to emulate the imaginary ideal world (in which we assume an incorruptible trusted mix) in the real world (without such an TTP) through an SMC protocol that is carried out by a set of dedicated mixing peers. In this way, we aim for a system that has the advantages of centralized mixing services with respect to high anonymity levels and deniability against outsiders without their disadvantages in terms of missing correctness and anonymity guarantees against insiders.

Our idea of modeling mixing as an SMC brings several advantages on a conceptual level: First, SMC directly guarantees the correctness of the computation which allows us to get rid of the disadvantageous group transaction pattern that is required in prior works [Max13a, RMSK14] to protect against theft and protocol aborts. Thereby, we overcome inherent restrictions of existing centralized and decentralized approaches with regards to anonymity and deniability while providing cryptographic guarantees of correctness in the seasoned SMC model. Second, SMC allows us to logically and physically separate users and mixing peers. This separation achieves better scalability and usability than previous decentralized approaches as well as higher failure tolerance and stronger resilience against protocol-level DoS attacks.

For the technical realization of such an SMC-based mixing service, we introduce a set of mixing peers that, in a distributed yet secure fashion, carry out the mixing in multiple standard one-to-one Bitcoin transactions. Using multiple sequential transactions instead of one atomic group transaction, the key challenge is to ensure that all mixing transactions succeed even when a fraction of the users or mixing peers fail or behave maliciously. To this end, we design an SMC protocol for the generation of Bitcoin transactions that is secure in the malicious adversary model. At the core of this protocol, we adopt and improve an existing threshold ECDSA signature scheme [IAIE03] to distributedly generate escrow addresses from which funds can only be redeemed via a threshold transaction, i.e., only when a majority of the controlling peers agrees to do so. Our proposed threshold transaction protocol is useful beyond our concrete use case, e.g., to secure digital wallets [BFK 15, GBF 14].

A second major challenge in our approach is the need to obliviously shuffle the output addresses of the participating users such that nobody, not even the mixing peers themselves, are able to trace or forge the shuffle. It is particularly important and difficult not to involve users in the shuffle (but unavoidable in prior works [Max13a, RMSK14]) in order to preserve the advantageous logical separation of users and mixing peers as argued above. To meet these challenges, we contribute the design and implementation of a novel efficient oblivious shuffle protocol, which is carried out only by the mixing peers and succeeds even if a subset of them actively tries to cheat. Our shuffle protocol, as well, is useful beyond our concrete use case, e.g., privacy preserving data collection [BS06, ZHBW17] or anonymous group messaging [CGF10].

Based on our improved and novel building blocks, we implement a functionally complete prototype of CoinParty to validate the claimed advantages of our approach. The discussion and evaluation of our prototype shows that CoinParty indeed realizes the advantages of both the early centralized approaches (i.e., large anonymity sets and deniability based on the property of mix indistinguishability) and the later decentralized approaches (i.e., correctness and anonymity guarantees against insid-
3.3 Decentralized Mixing of Digital Currencies

In this section, we present the design of CoinParty, a novel mixing service that fulfills the requirements for ideal mixing services stated in Section 3.2.3. On the highest level, our idea is to realize the insecure centralized mixing model securely in a decentralized fashion via SMC (Section 3.3.1): In the centralized model, users deliver their funds in escrow to the mixing service and are paid back at a later point...
Decentralized Mixing of Digital Currencies

3. Decentralized Mixing of Digital Currencies

In CoinParty (Section 3.3.2), a set of mixing peers emulates the centralized mixing service through a distributed cryptographic protocol, thereby obviating the need for a TTP and protecting against theft from the mixing service itself. This approach allows us to combine the advantages of previous centralized and decentralized approaches in one system, but challenges us with the design of corresponding building blocks (Section 3.3.3) and SMC protocols (Sections 3.3.4 to 3.3.8) that efficiently and securely compute the desired mixing.

3.3.1 Mixing as an SMC problem

During our analysis of previous mixing systems (Section 3.2.4), we have found different shortcomings of prior works with respect to their security and anonymity guarantees. Many of these are rooted in the use of corruptible central third parties or in the insecure design of distributed mixing protocols. These observations motivate us to model the mixing problem as an SMC.

To this end, we first have to define mixing in an ideal world assuming an incorruptible third party and must then analyze whether this ideal model suffices our needs. After selecting an adequate adversary model, the broad base of established feasibility results yields the exact security guarantees that we can achieve in the real world. With this theoretical model in place, designing our actual system then boils down to the non-trivial task of replacing the TTP in the ideal world with a sufficiently efficient SMC protocol that realizes the same functionality securely in the real world.

Mixing in the Ideal World

We consider \( n \) participating users \( U_1, \ldots, U_n \) who want to securely carry out a mixing, denoted by function \( F_{Mix}(\cdot) \). The private input of \( U_i \) to \( F_{Mix}(\cdot) \) is the fresh output address \( O_i \). The second input (not necessarily private) is the required amount of bitcoins \( \nu \) in the form of an accepted transaction \( I_i \rightarrow E_i \) (referred to as the commitment \( C_i \)) to an escrow address \( E_i \) under control of the mixing service. The desired outputs are \( n \) transactions \( E_i \rightarrow O_{\pi(i)} \) (referred to as refunds \( R_i \)), where \( \pi \) is a random and secret permutation.

In the imaginary ideal world, where an incorruptible TTP exists, mixing proceeds as follows: The TTP creates \( n \) fresh addresses \( E_i \) and instructs each user \( U_i \) to commit \( \nu \) to \( E_i \). The TTP waits until the Bitcoin network has confirmed these commitments (i.e., when six subsequent blocks have been mined [Ros14]). The TTP then draws a random permutation \( \pi \) and creates and signs the \( n \) refunds \( R_i : E_i \rightarrow O_{\pi(i)} \). Finally, the valid signed transactions are returned to the mixing participants or announced directly to the Bitcoin network.

Security of Mixing in the Ideal World

All security and privacy requirements listed in Section 3.2.3 hold in the ideal execution: The third party always produces a correct mixing in a timely manner and never reveals which user submitted which output address nor the random permutation used for the mixing. The third party also only creates a single valid transaction
from each address $E_i$ such that double spending is ruled out and the returned transaction $R_i : E_i \rightarrow O_{\pi(i)}$ will be accepted by the Bitcoin network.

We emphasize that the commitments $C_i$ are only due to the problem of double spending in Bitcoin [Nak08] and do not arise from our use of SMC. To make this clear, we consider to take the private keys corresponding to the input addresses $I_i$ as inputs to the secure computation and directly output transactions $R_i : I_i \rightarrow O_{\pi(i)}$. From an SMC point of view this does not present a problem since SMC’s privacy guarantee makes sure that the private keys are not learned by anyone. However, due to the way the Bitcoin network accepts transactions, the following scenario then becomes possible: It might occur that user $U_i$ has already been refunded through transaction $R_j : E_j \rightarrow O_{\pi(j)}$, but $R_i : E_i \rightarrow O_{\pi(i)}$ has not been issued, i.e., $U_i$’s own bitcoins have not been spent yet. $U_i$ can then try to keep her funds by announcing $R_i' : I_i \rightarrow O_{\pi(i)}$ before or simultaneously with $R_i$. This equivocation results in an incorrect mixing in which user $U_{\pi(i)}$ is not refunded when the Bitcoin network decides to accept $R_i'$ and to reject $R_i$. This could be prevented in SMC but this would require for all Bitcoin miners to participate in the mixing protocol since they decide which transactions are included in the blockchain. As this is clearly infeasible, the commitments are inevitable in our SMC mixing model. Note that all centralized mixes (which we set out to emulate securely in cryptography) require commitments for the same reason.

Choice of the Adversary Model and Further Assumptions

Having defined the ideal mixing model, we still need to determine an adversary model to work with and realistically capture the threats in the real world. As detailed in Section 2.2, we have to consider passive versus active, static versus adaptive, and bounded versus unbounded adversaries.

Due to the monetary value involved in Bitcoin and the inherently trustless and anonymous environment (e.g., there is no possibility to define contracts between mixing participants to ensure semi-honest behavior), we argue that we need to consider active adversaries who will try to actively cheat participants out of their money or sabotage competing mixing services. In this vein, we also defensively choose the static corruption strategy which assumes that peers are corrupted from the start. Note that this is a more pessimistic assumption than an adaptive adversary who corrupts peers only later during the computation. Finally, since the whole security of Bitcoin depends on computational assumptions (i.e., the hardness of computing discrete logarithms and proof-of-works), we also assume a computationally bounded adversary – an unbounded adversary would not need to attack our mixing but could directly steal bitcoins from users by brute-forcing their private keys.

Theoretical Feasibility Results

We briefly review the established feasibility results for SMC [LP09] to characterize what security guarantees we can achieve in the chosen adversary model. Given $m$ peers of which at most $\tau$ are statically corrupted by a malicious adversary, the following thresholds have been proven as tight upper bounds. For $\tau < m/3$, SMC protocols with full privacy, correctness, and robustness are possible, e.g., [GMW87,
Figure 3.3 Overview of CoinParty with three mixing peers and four users. Mixing peers initialize a mixing operation by creating a sufficient amount of escrow addresses (Sec. 3.3.4). The users commit their funds to these escrow addresses and provide fresh unlinkable output addresses in an encrypted manner (Sec. 3.3.5). Mixing peers form a decryption mixnet to obliviously shuffle the provided output addresses (Sec. 3.3.6). Finally, the funds are transferred from the escrow to the unlinkable output addresses (Sec. 3.3.7). If an error occurs, e.g., due to malicious peers, the mixnet recovers and determines the misbehaving peers (Sec. 3.3.8).

BGW88]. For $\tau < m/2$, SMC protocols with full privacy, correctness, and robustness can be achieved assuming access to a broadcast channel, e.g., [GMW87,RB89]. For $\tau \geq m/2$, protocols with privacy and correctness but without robustness can be achieved, e.g., [DKL+13]. Since, robustness is a crucial property for CoinParty as argued above and we realistically assume that no broadcast channel but only private point-to-point channels are available, e.g., realized using Transport Layer Security (TLS), we can only tolerate at most $\tau < m/3$ corrupted peers.

We emphasize that these feasibility results for SMC provide provable upper limits on how many and what type of adversaries we can tolerate during mixing. In consequence, any mixing protocol that claims better security must either make weaker assumptions or operate outside the SMC model and should thus be treated with care.

For example, Bitcoin’s group transactions (i.e., the basic pattern underlying related works) is commonly claimed to be secure against $n-1$ corrupted peers. However, this notion of security depends on the honest behavior of Bitcoin miners and is not provable in the rigorous SMC model underlying our approach.

We now design a practical mixing system that is secure against $\tau < m/3$ corrupted peers. According to the summarized feasibility results, this threshold is optimal for malicious adversaries and point-to-point communication which are defensive yet realistic assumptions in the context of mixing real monetary values such as bitcoins.

### 3.3.2 System Overview

We provide an overview of our secure distributed mixing protocol, CoinParty, which is carried out by a network of mixing peers to emulate the ideal mixing model in the real world. In our approach, users can be mixing peers themselves (e.g., in a local mixing service set up among a group of friends) or users and mixing peers can be logically and physically separated (e.g., in a commercial distributed mixing service...
The initial phase (Section 3.3.4) involves only the mixing peers and can take place at any time before an actual mixing operation. It is used to precompute expensive parts of the later phases, most importantly a set of escrow addresses $E_1, \ldots, E_n$. These escrow addresses are standard Bitcoin addresses that are under the shared control of the mixing peers.

During the subsequent commitment phase (Section 3.3.5), the mixnet accepts participants for the upcoming mixing operation. Notably, this is the only phase in which users need to interact with the mixnet. A user Alice, who wants to join a mixing operation, simply sends a request to any mixing peer and receives an individual escrow address ($E_1$ in Figure 3.3) together with a set of parameters and conditions for the mixing such as the required amount $\nu$. If Alice agrees to these conditions, she commits the required funds to the escrow address by simply issuing a standard Bitcoin transaction over $\nu$ to $E_1$. Alice also generates a fresh output address $O_1$ and uses the $m$ public keys of the mixing peers to create a layered encryption of $O_1$ that she sends to the mixing peers.

After a certain time and a minimum number of committed users, the window for participation is closed and the shuffle phase (Section 3.3.6) starts. One after the other, each mixing peer $M_i$ decrypts the outmost layer of the encrypted output addresses, secretly shuffles the still encrypted addresses, and forwards them to the next mixing peer. At the end of the shuffle phase, mixing peers obtain a random shuffle $O_{\pi(1)}, \ldots, O_{\pi(n)}$ of the users’ output addresses. Notably, the permutation $\pi$ of the addresses is unknown to the mixing peers themselves, thus users and their output addresses are unlinkable even to them.

In the final transaction phase (Section 3.3.7), the funds committed to the escrow addresses are transferred to the shuffled output addresses in $n$ standard Bitcoin transactions $E_i \rightarrow T_i$. Transactions are signed with a threshold signature scheme which ensures that all transactions succeed and that malicious mixing peers or other attackers cannot steal funds or halt the transaction phase prematurely.

As we discuss in detail in Section 3.4.1, the initialization, commitment, and transaction phases succeed even in the presence of up to $\tau < m/3$ malicious mixing peers and any number of malicious users. However, a single malicious user may cause the shuffle phase to fail by announcing incorrect information during the commitment phase – we emphasize that this is also possible in all other decentralized mixing approaches [Max13a, Ros12, RMSK14]. In this case, a fifth phase, recover and blame (Section 3.3.8), is entered which identifies and punishes the misbehaving users and transfers funds back to the participating users without shuffling. Notably, our design directly allows mixing peers to retain funds of malicious users thereby providing at least reactive protection against DoS attacks from malicious users. This is not possible in the previous decentralized approaches that are based on atomic group transactions [Max13a, Ros12, RMSK14].

We now briefly explain the required cryptographic building blocks that were not covered in the background chapter, then explain in detail each of the five phases and present the corresponding secure protocols.
3. Decentralized Mixing of Digital Currencies

3.3.3 Cryptographic Building Blocks

The core building block of CoinParty is a threshold ECDSA scheme secure against $\tau < m/3$ malicious peers. This scheme is used to generate escrow addresses (Section 3.3.5) and to issue threshold transactions (Section 3.3.7). Note that all threshold cryptography schemes are just special cases of SMC and can thus be realized using generic SMC protocols. We thus extend the background given in Section 2.3.3.1 and show how to realize maliciously secure SMC protocols using Verifiable Secret Sharing (VSS). We then describe how efficient threshold ECDSA key generation and signature schemes can be realized based on these techniques. VSS is also used to verify the integrity of our shuffle protocol (Section 3.3.6).

Generic SMC via Secret Sharing in the Malicious Model

Maliciously secure SMC based on secret sharing proceeds as outlined in Section 2.3.3.1: The desired functionality $F(\cdot)$ is first expressed as an arithmetic circuit $F_{\text{Arith}}(\cdot)$ of addition and multiplication gates, which is subsequently evaluated by the peers. Each peer first secret-shares its private input to the other peers. Together, the peers then carry out the required additions and multiplications on the secret shares, i.e., without learning the secret inputs. At the end, the output is obtained in secret-shared form and can be recombined.

Sharing: We denote $\langle x \rangle \leftarrow \text{SHARE}(x, t, m)$ the sharing of a secret $x$ across $m$ peers such that nothing can be learned from any subset of less than $t$ shares. In the semi-honest model, we simply draw and distribute $m$ points from a random polynomial (cf. Section 2.3.3.1). In the malicious adversary model, we additionally need to make sure that a malicious peer cannot equivocate, i.e., send inconsistent shares to the other peers. This is usually achieved through commitments, e.g., in Pedersen’s VSS scheme [Ped92], which we employ for our protocols. Using Pedersen’s VSS scheme, the SHARE operation is secure for any $\tau < t \leq m$ malicious peers.

Addition: $\langle a + b \rangle \leftarrow \text{ADD}(\langle x \rangle, \langle y \rangle)$ computes the sum of two secret-shared values $x$ and $y$ revealing neither the values nor the result. In Shamir’s and any other additive secret sharing scheme, ADD can be computed locally by each peer (cf. Section 2.3.3.1). Pedersen’s VSS scheme is also linear and ADD has the same security as SHARE, i.e., it is robust against $\tau < m$ malicious peers [Ped92]. Remember that we use the shorthand notation $\langle x + y \rangle = \langle x \rangle \oplus \langle y \rangle$ as introduced in Section 2.3.4.

Multiplication: $\langle x \cdot y \rangle \leftarrow \text{MULTIPLY}(\langle x \rangle, \langle y \rangle)$ denotes the multiplication of two secret-shared values $x$ and $y$ and is realized using an interactive protocol [BGW88], which is secure against $\tau < m/2$ passively and $\tau < m/4$ actively corrupted peers. A protocol that is secure against $\tau < m/3$ actively corrupted peers has been described and fully proven in [ALR11]. We use the shorthand notation $\langle x \cdot y \rangle = \langle x \rangle \odot \langle y \rangle$ as introduced in Section 2.3.4.
Recombination: We denote by \( x \leftarrow \text{RECOMBINE}(\langle x \rangle) \) the reconstruction of the secret \( x \) from any set of at least \( t \) shares, i.e., the dual operation to \text{SHARE}. In the semi-honest case, recombination can be done using Lagrange interpolation (cf. Section 2.3.3.1). In the malicious model, we need to filter out inconsistent shares, e.g., those sent by malicious users to forge recombined secrets. Interpreting the shares \( \langle x \rangle \) as a Reed-Solomon code [RS60] for \( x \) as proposed by [MS81] and then using the Welch-Berlekamp algorithm [WB86], we can correct up to \( m/3 - 1 \) erroneous shares. Therefore, \text{RECOMBINE} is secure against \( \tau < m/3 \) malicious peers.

For the remainder of this chapter, we always use the maliciously secure primitives if not explicitly stated otherwise. We now proceed to construct a maliciously secure threshold ECDSA scheme from these primitives.

**Threshold ECDSA**

The basic idea to prevent theft in CoinParty is to let users transfer funds to escrow addresses that are owned and controlled not by one but multiple mixing peers. After shuffling the funds, the mixing peers then need to collaborate to transfer funds back to the users which prevents cheating. To receive and spend funds from escrow addresses, we need two cryptographic primitives, i) a Distributed Key Generation (DKG) scheme for ECDSA keys and ii) a threshold ECDSA signature scheme.

Principally, we could use the construction for generic SMC presented in the previous section to implement these primitives. Indeed, [KSS12, KSMB13] implement 1024 bit Rivest-Shamir-Adleman (RSA) signature generation in a generic SMC framework. Although they use a representation as Boolean circuit instead of an arithmetic circuit and RSA instead of ECDSA, the reported runtime of more than 15 h for the generation of one signature clearly shows the need for more efficient custom protocols.

We briefly present the DKG and signature scheme that are used as building blocks for CoinParty. For both schemes, the domain parameters \( P = (q, (a, b), G, n) \) of an elliptic curve \( E_P \) are given: \( n \in \mathbb{N} \) is the order of the generator \( G \in E_P \) that generates the elliptic curve \( E_P = \{ (x, y) \in \mathbb{Z}_q^2 \mid y^2 = x^3 + ax + b \} \) with \( a, b \in \mathbb{Z}_q \). In the context of Bitcoin, the parameters of the standardized curve secp256k1 [Res00] are used.

**Distributed Key Generation.** We adapt the DKG scheme by Gennaro et al. [GJKR07] to use an elliptic curve as the underlying cyclic group, referred to as EcDkg in the following. We briefly describe the high-level ideas and refer to Protocol 3.1 and [GJKR07] for the details. The following steps are executed by each of the \( m \) peers: \( P_i \) chooses a random secret \( s_i \) and verifiably shares this secret to the other peers \( P_{j\neq i} \) (Steps 1 and 2). Any misbehaving peer is disqualified (Step 3). The private key \( d \) then corresponds to the sum of the secrets \( s_i \) of the remaining qualified peers \( Q \), i.e., \( d = \sum_{i \in Q} s_i \). Note that \( d \) is never explicitly computed by anyone. Instead, \( P_i \) computes its individual secret share \( \langle d \rangle_i \) of the private key \( d \) by summing up the shares of the random secrets \( s_j \) received in Step 1 from the other peers \( P_{j\neq i} \) (Step 4). To compute the public key \( D = dG \) without revealing \( d \), \( P_i \) broadcast its individual part \( S_i = s_iG \) and computes \( D = \sum_{j \in Q} S_j = \sum_{j \in Q} s_jG = (\sum_{i \in Q} s_i)G = dG \) using its own share \( S_i \) and
Then, PECDSA in standard results discussed in Section 3.3.1 which state that anything better than scheme [GJKR07] is thus reduced to
the shares

Protocol 3.1

**Input:** Generator G of Elliptic Curve ℘, point H ∈ ℘

**Output:** Share of private key \( \langle d_i \rangle \), public key \( D = dG \)

1. Choose and share secret \( s_i \) and nonce \( r_i \) using random polynomials \( S_i(x) = \sum_{k=1}^{t} s_{i,k} x^k + r_i \) and \( R_i(x) = \sum_{k=1}^{t} r_{i,k} x^k + r_i \).
2. Broadcast \( C_{i,k} = s_{i,k} G + r_{i,k} H \) for \( k = 1..t \).
3. Check \( \langle s_j \rangle G + \langle r_j \rangle H = \sum_{k=1}^{t} i C_{j,k} \).
   If check fails, broadcast a complaint against \( P_j \) and disqualify any \( P_j \) that was complained against at least \( t + 1 \) times.
4. Compute share \( \langle d_i \rangle = \bigoplus_{j \in \mathcal{Q}} \langle s_j \rangle \) of the private key, with \( \mathcal{Q} \) the set of qualified peers.
5. Broadcast \( S_{i,k} = s_{i,k} G \) for \( 0 \leq k \leq t \).
6. Check \( \langle s_j \rangle G = \sum_{k=1}^{t} i S_{j,k} \) for \( 1 \leq j \neq i \). If the \( j \)th check fails, broadcast \( \langle s_j \rangle, \langle r_j \rangle \).
   If valid complaints against \( P_j \) are received, reconstruct \( s_j \) and compute \( S_{j,0} \).
7. Compute public key \( D = \sum_{j \in \mathcal{Q}} S_{j,0} \).

**Protocol 3.1** The EC_DKG scheme as executed by a peer \( P_i \) based on Gennaro et al. [GJKR07].

the shares \( S_j \) received from \( P_j \) (Steps 5 and 7). A second round of commitments and checks protects against malicious adversaries (Step 6).

As proved in [GJKR07], EC_DKG is secure against \( \tau < m/2 \) semi-honest and malicious peers. It is important to note that this result depends on the availability of a reliable broadcast channel. In the Internet, one can usually only assume private point-to-point channels, e.g., realized using TLS in our case. Different protocols exist to implement reliable broadcast over such private channels but these only achieve a security threshold of \( \tau < m/3 \). In our setting, the security of Gennaro’s DKG scheme [GJKR07] is thus reduced to \( \tau < m/3 \). This is consistent with the feasibility results discussed in Section 3.3.1 which state that anything better than \( \tau < m/3 \) is impossible without a reliable broadcast channel.

**Threshold Signatures.** To achieve threshold ECDSA signatures, we adopt the scheme by Ibrahim et al. [IAIE03] as detailed in Protocol 3.2. First, peers run the previously described EC_DKG scheme (Step 1) and \( P_i \) interprets the output as a share of a random nonce \( k \) and the corresponding point \( kG \) on the elliptic curve \( \mathcal{E}_\mathcal{P} \).

The \( x \)-coordinate of the point \( kG \) defines the first part \( r \) of the signature.

Then, \( P_i \) inverts \( \ell \), using the INVERT protocol given in [IAIE03] (Step 2).

As in standard ECDSA [Nat94], \( P_i \) builds and truncates the hash of the message to be signed (Step 3). Finally, \( P_i \) computes and recombines the second part \( s \) of the signature (Steps 4 and 5).

Ibrahim et al. [IAIE03] show that their protocol is secure against \( \tau < m/2 \) semi-honest but only \( \tau < m/4 \) malicious peers. This is due to their employed MULITPLY primitive which is only secure against an \( \tau < m/4 \) adversary. Using the MULITPLY primitive described above which provides security against \( \tau < m/3 \) malicious peers, TRESHOLD_ECDSA becomes secure against \( \tau < m/3 \) malicious peers.

With the maliciously secure EC_DKG and TRESHOLD_ECDSA schemes, we now have all building blocks to construct the secure protocols for the five phases of CoinParty, i.e., initialization, commitment, shuffle, transaction, and the optional blame-and-recover phase.
Input: Generator $G \in \mathcal{E}_P$, point $H \in \mathcal{R}_E$, message $m$, share of private key $\langle d \rangle_i$

Output: ECDSA signature $(r, s)$

1. Compute $\langle k \rangle_i, kG \leftarrow \text{EcDkg}(G, H)$. With $(x, y) = kG$ repeat until $x \neq 0$. Set $r = x$.

2. Compute $\langle k^{-1} \rangle_i \leftarrow \text{Invert}(\langle k \rangle_i)$.

3. Hash message $e \leftarrow \text{Hash}(m_{|0:n|})$.

4. Compute $\langle s \rangle_i = \langle k^{-1} \rangle_i \odot (e \oplus r \langle d \rangle_i)$.

5. $s \leftarrow \text{Recombine}(\langle s \rangle_i)$ and repeat 1) - 5) until $s \neq 0$.

6. Return signature $(r, s)$.

Subprotocol $\text{Invert}(\langle k \rangle_i, G, H)$

1.1 Draw random blind $\langle r \rangle_i \leftarrow \text{EcDkg}(G, H)$.

1.2 Blind share multiplicatively $\langle u \rangle_i = \langle k \rangle_i \odot \langle r \rangle_i$.

1.3 Recombine and invert $u^{-1} \leftarrow \text{Recombine}(\langle u \rangle_i)^{-1}$.

1.4 Return $\langle k^{-1} \rangle_i = u^{-1} \langle r \rangle_i$.

Protocol 3.2 The ThresholdECDSA signature scheme due to Ibrahim et al. [IAIE03] as executed by $P_i$.

3.3.4 The Initialization Phase

The initialization phase is executed before a new mixing operation starts. It has three tasks, i) set up and share session key pairs, ii) generate escrow addresses, and iii) precompute partial signatures for future transactions to redeem funds from these addresses. This phase does not require any interaction with the future participants and can thus be executed at any time before the actual mixing operation in order to speed up the subsequent protocol phases.

We assume that mixing peers $M_1, \ldots, M_m$ are already fully connected to each other and have established private channels, e.g., using TLS. We do not explicitly consider how a mixnet would be bootstrapped. In [Max13a, RMSK14], users bootstrap the mixnet among each other in an ad hoc fashion, which is also easily possible in CoinParty. Another approach is to maintain public directories of mixing peers, e.g., similar to node discovery in Tor [DMS04].

We further assume that the mixing peers have agreed on a set of parameters $\mathcal{P} = (n_{\min}, n_{\max}, \nu, t_{\infty}, \omega)$ and an elliptic curve $\mathcal{E}_{P^2}$. $n_{\min}$ and $n_{\max}$ are lower and upper bounds on the number of participants. The mixing value $\nu$ denotes the amount of bitcoins each user has to commit by the deadline $t_{\infty}$. $\omega$ denotes the time window allocated for the refunds, i.e., users are repaid at latest by $t_{\infty} + \omega$. As we show in Section 3.4.2, anonymity increases with the length of the mixing window $\omega$.

Protocol 3.3 shows the detailed initialization protocol executed by each mixing peer $M_i$. First, the mixing peers generate one ECDSA key pair per peer that is used as a session key (Step 1). Note that only $M_i$ learns the private key $sk_i$ in clear, but all other mixing peers $M_j$ hold a share $\langle sk \rangle_j$ and know the public key $K_i$. When an error occurs in later protocol phases, the mixing peers use their shares to reconstruct $sk_i$ in order to check whether $M_i$ misbehaved.

\footnote{In Bitcoin, the standardized curve secp256k1 [Res00] is used.}
Input: Mixing parameters $P = (n_{\text{min}}, n_{\text{max}}, \nu, t_{\text{in}}, \omega)$, generator $G \in \mathbb{G}$, $H \in \mathbb{G}$

Output: Escrow addresses $(\langle d_{1j} \rangle, E_1), \ldots, (\langle d_{n_{\text{max}}j} \rangle, E_{n_{\text{max}}})$

1. Set up session keys
   
   (a) $(sk_j)_i, K_j \leftarrow \text{EcDkg}(G, H)$ for $j = 1 \ldots m$.
   
   (b) Send $(sk_j)_i$ to $M_{j=1 \ldots m}$ and receive $(sk_i)_j$ from $M_{j=1 \ldots m}$.
   
   (c) $sk_i \leftarrow \text{Recombine}(sk_j)$

2. Generate escrow addresses $E_{j=1 \ldots n_{\text{max}}}$
   
   (a) Generate keys $(\langle d_j \rangle, d_jG) \leftarrow \text{EcDkg}(G, H)$, $j = 1 \ldots n_{\text{max}}$.
   
   (b) Hash public keys $d_jG$ to Bitcoin addresses $E_j$.

3. Precompute $\text{ThresholdEcdsa}$ for each escrow address $E_{j=1 \ldots n_{\text{max}}}$
   
   (a) Generate nonce $[k_j], k_jG \leftarrow \text{EcDkg}(G, H)$. Set $(x, y) = k_jG, r = x$. Repeat if $r = 0$.
   
   (b) Invert nonce $(k_j^{-1}) \leftarrow \text{Invert}([k_j])$.
   
   (c) Compute partial signature $(s_j')_i = (k_j^{-1})_i \odot r(d_j)_i$.

Protocol 3.3 The Initialization protocol as executed by $M_i$. Mixing peers set up session keys which are secret-shared among each other. They then generate escrow addresses and precompute partial signatures.

The second part of Protocol 3.3 is dedicated to generating escrow addresses (Step 2). First, the mixing peers generate $n_{\text{max}}$ key pairs (using EcDkg) and transform the public keys into Bitcoin addresses $E_j$ according to the specifications of the Bitcoin protocol [Nak08]. The addresses $E_j$ are used as escrow addresses in the subsequent commitment phase, i.e., participants of the mixing need to transfer their funds to these addresses before the mixing starts.

Since the private keys $d_j$ corresponding to the addresses $E_j$ are shared among the mixing peers, they can only collaboratively redeem funds from escrow addresses via threshold signatures. In the third part of the initialization protocol, the mixing peers partially precompute one threshold signature for each escrow address $E_j$ (Step 3). The signature cannot be fully precomputed since it is not yet known to which output address the funds escrowed at $E_j$ need to be transferred. This is only determined at the end of the shuffle phase (Section 3.3.6).

3.3.5 The Commitment Phase

After the mixing operation has been initialized (Section 3.3.4), the mixing network enters the commitment phase and starts to accept participants. The goal of this phase is to have at least $n_{\text{min}}$ and at most $n_{\text{max}}$ users commit the required $\nu$ bitcoins to one of the previously computed escrow addresses before the deadline $t_{\text{in}}$. The commitment phase is the only phase that requires a user to interact with the mixing peers. A user only needs a way to transfer bitcoins (e.g., using the standard Bitcoin client or a web-based wallet) and a web browser to participate, which makes CoinParty accessible to a broad user base as discussed further in Section 3.4.6.
Input: Mixing parameters $\mathcal{P}$, escrow addresses $E_1, \ldots, E_n$.
Output: Escrow addresses $E_1, \ldots, E_n$ that have been committed to.

Bootstrap:
1. $U_j$ chooses any mixing peer $M_i$ as entry peer and sends a request for participation.
2. $M_i$ assigns $E_i$ to $U_j$ and announces the choice to $M_{k=1}^n$.
3. $M_i$ sends to $U_j$ session ID $S_{ij}$, parameters $\mathcal{P}$, escrow address $E_j$, identities $(M_1, \ldots, M_n)$, public keys $(K_1, \ldots, K_n)$.
4. $U_j$ contacts $M_i, \ldots, M_m$ to verify $K_1, \ldots, K_m$ to check consistency of $\mathcal{P}$ and $E_j$ for $S_{ij}$.

Commit:
5. $U_j$ generates a fresh unlinkable Bitcoin address $O_j$.
6. $U_j$ encrypts $O_j$ in layers, i.e., $[O_j]_{K_1, K_m} = E_{K_1}([E_{K_1}([\ldots E_{K_n}(O_j)])])$.
7. $U_j$ computes $(C^j) \leftarrow \text{Share}(\text{Hash}(O_j), K_1, \ldots, K_m)$ and $(C^m_{j+1}) \leftarrow \text{Share}(\text{Hash}(O_j))$.
8. $U_j$ broadcasts $(S_{ij}, [O_j]_{K_1, K_m}, (C^j), (C^m_{j+1}))$ to $M_{k=1}^m$.
9. $U_j$ commits $O_j$ to $E_j$ in a transaction $I_j \to E_j$.
10. $M_{k=1}^m$ verifies that $I_j \to E_j$ has been accepted by the Bitcoin network.

Protocol 3.4: The COMMITMENT protocol: A user $U_j$ who wants to join the mixing first bootstraps to the mixing network, using any mixing peer $M_i$ as entry point. After verifying parameters $\mathcal{P}$ and the assigned escrow address $E_j$, the user announces her encrypted output address $O_j$ and secret-shares checksums $C^j$. Finally, the user transfers the required amount of $nu$ bitcoins to the assigned escrow address $E_j$ and thereby fully commits to the mixing.

Protocol 3.4 shows the detailed steps of the commitment phase. In order to connect to the mixing network, user $U_j$ chooses any mixing peer $M_i$ as entry point (Step 1). $M_i$ selects an unused escrow address $E_i$ from the pool of precomputed addresses and announces its choice to all other mixing peers (Step 2). Then, $U_j$ is provided a session ID $S_{ij}$, the parameters of the mixing $\mathcal{P}$, the escrow address $E_j$, as well as the identities and public keys of the other mixing peers. Since $U_j$ could potentially have chosen a malicious mixing peer as entry point, it needs to verify the received information: $U_j$ simply contacts and authenticates the other mixing peers $M_{k=1}^m$ by their public keys $K_k$ and presents $S_{ij}$ upon which each mixing peer responds with $\mathcal{P}$ and $E_j$ (Step 4). If $U_j$ receives ambiguous information, $U_j$ aborts the mixing and sends a report of the equivocation to the mixing network. Mixing peers can then collaboratively single out misbehaving peers.

When the user $U_j$ has validated $\mathcal{P}$ and $E_j$, $U_j$ proceeds with the commitment. First, $U_j$ generates a fresh unlinkable output address $O_j$ and encrypts it in $m$ layers using a semantically secure public key encryption scheme, e.g., the Elliptic Curve Integrated Encryption Scheme (ECIES) (Steps 5 and 6). This layered encryption prevents mixing peers from learning which user submitted which output address and will be lifted one after the other in the subsequent shuffle phase (Section 3.3.6).

Second, to prevent a malicious mixing peer from substituting $U_j$’s output address with its own, e.g., to steal funds, $U_j$ additionally provides verification information that enables the honest mixing peers to verify the integrity of the shuffle at each layer (Steps 7 and 8). In particular, $U_j$ hashes each encryption layer of her output address $O_j$ using a cryptographic hash function, i.e., for each layer $l = 1 \ldots m$, $U_j$
User $U_j$ creates a layered encryption of her output address $O_j$ and computes a checksum $C_j^l$ at each layer $l = 1, \ldots, m+1$ which is secret-shared to the mixing peers to enable verification of the shuffle.

Finally, $U_j$ transfers the required $\nu B$ to $E_j$ (Step 9). After this point, $U_j$ is fully committed to the mixing operation and no further interaction with $U_j$ is required. The commitment phase is completed when $U_j$’s commitment $I_j \rightarrow E_j$ has been accepted and confirmed by the Bitcoin network (Step 10).

### The Shuffle Phase

The goal of the shuffle phase is to obliviously and verifiably permute the output addresses of the participating users. A shuffle is oblivious if no one, not even the mixing peers know the applied permutation, i.e., the link between a user and her fresh output address. Verifiability means that the shuffled items cannot be tampered with, i.e., no output address is deleted, substituted, added, or modified.

Decryption mixnets, as proposed by Chaum [Cha88], are one solution to realize oblivious and verifiable shuffles. They have been successfully applied, e.g., to anonymous communications [BS06, CGF10] and Bitcoin mixing [RMSK14]. Our shuffle protocol is also based on decryption mixnets but takes a fundamentally different approach to ensure verifiability, which improves performance and enables participants to plausibly deny having participated. We briefly present our shuffle protocol first and then compare our protocol to the related proposals of [BS06, CGF10, RMSK14].

Protocol 3.5 shows the detailed shuffle protocol executed by the mixing peers. The input (i.e., the encrypted output addresses and checksums) has already been provided by the users during the previous commitment phase (Section 3.3.5). Now, each mixing peer $M_i$, one after the other, decrypts one layer of the encrypted output addresses (Step 1a), shuffles them (Step 1b), and broadcasts the result. Before proceeding, the other mixing peers need to verify that $M_i$ indeed decrypted correctly and did not tamper with the (encrypted) output addresses. To this end, they securely aggregate the provided checksums, recombine the result, and compare it to the checksum computed over the decryptions obtained from $M_i$ (Step 1c). Finally, after $m$ rounds of decryption and shuffling, the output addresses $O_1, \ldots, O_n$ are obtained in the clear but in an unknown order since they are already shuffled under...
### 3.3. Decentralized Mixing of Digital Currencies

**Input:** Layered encryptions $[O_j]_{K_i}^{r_i} \cdot j = 1 \ldots n$, shared checksums $(C^l_j), \ j = 1 \ldots n, l = 1 \ldots m+1$

**Output:** A shuffle $S = (O_{x(1)}, \ldots O_{x(n)})$ under random and secret permutation $\pi$

1. Repeat for each $M_{x(i)}$:
   (a) $M_i$ receives $S' = ([O_{x(i)}, \ldots O_{x(i)}]_{K_i}^{r_i})$ and removes the outermost encryption by applying $Dec_{K_i}$ to each component.
   (b) $M_i$ draws a secret permutation $\pi_i$ and broadcasts $S'^{\pi_i} = ([O_{x(i)}, \ldots O_{x(i)}]_{K_i}^{r_i})_{\pi_i}^{r_i}$.
   (c) All mixing peers $M_{x(i)}$ collaboratively check that $M_i$ decrypted correctly
      - Compute checksum $C_{i+1} = \text{RECOMBINE}(C_{i+1})$ with $(\sum_{j=i}^{n} C_{i+1}^l)$.
      - Compute checksum $C_{i+1} = \sum_{j=i}^{n} \text{Hash}([O_{x(i)}, \ldots O_{x(i)}]_{K_i}^{r_i}).$
      - If $C_{i+1} \neq C_{i+1}^{\pi_i}$ invoke $\text{BLAMEANDRECOVER}($,$,^{\pi_i}($,$,^{\pi_i}))$.

2. All mixing peers conduct final shuffle:
   (a) Sort output addresses lexicographically, i.e., $S^{\pi} \leftarrow \pi_{rand}(S^{m+1})$.
   (b) Seed PRNG with $C = \sum_{i=1}^{m+1} C_i$, draw $\pi_{rand} \leftarrow \text{PRNG}$, and apply to $S^{\pi}$.
   (c) Output $S = \pi_{rand}(S^{\pi}) = (O_{x(1)}, \ldots O_{x(n)})$ with $\pi = \pi_{rand} \circ \pi_{lex} \circ \pi_{n} \circ \ldots \circ \pi_{1}$.

**Protocol 3.3** The *Shuffle* protocol: Users provide layered encryptions of their output addresses and secret-shared checksums. Mixing peers form a mixing network to shuffle these addresses obliviously and check correctness after each step. A final fixed ordering combined with a common but random permutation ensures randomness of the shuffle even in the presence of malicious mixing peers.

The permutation $\pi_{n} \circ \ldots \circ \pi_{1}$. Note that after Step 1, any link between a user and her output address has thus been destroyed as long as at least two honest mixing peers have secretly and randomly shuffled the encrypted addresses in Step 1b. The mixing peers proceed to sort the addresses lexicographically (Step 2a). They then seed a pseudorandom number generator (PRNG) from the checksums and draw a common random permutation $\pi_{rand}$ from it (Step 2b) and obtain the final shuffle $S$ (Step 2c). Steps 2a, 2b, and 2c are important to guarantee the randomness of the shuffle – without these steps the last mixing peers could dictate the final shuffle.

Our protocol improves over [RMSK14, BS06, CGF10] in two ways: First, we use secret-shared checksums to efficiently validate the correctness of the shuffle on each stage (Step 1c). Using these checksums, mixing peers can verify the shuffle at each stage without involving the users. The advantages of this approach compared to the protocols proposed in [RMSK14, BS06, CGF10] are twofold. First, tampering is detected immediately and not only after the shuffle has finished incorrectly. Thus, less processing and communication resources are wasted on incorrect shuffles. Second, a user does not learn which other users participate in the shuffle, which increases anonymity (cf. Section 3.4.2) and enables deniability (cf. Section 3.4.3).

As a second improvement, we fix the following design flaw in the shuffle protocol of [RMSK14]. Similar to CoinParty, [RMSK14] relies on decryption mixnets to shuffle output addresses $O_1, \ldots, O_n$. After the first $m - 1$ shuffle steps in [RMSK14], the last mixing peer $M_m$ receives from $M_{m-1}$ the shuffle $S^m$ with only one layer of encryption left, i.e., $Enc_{K_m}$. Now, $M_m$ can first lift $Enc_{K_m}$ and then, with knowledge of the output addresses, apply a final permutation $\pi_m$. By choosing $\pi_m$ accordingly, $M_m$ can thus select which output address receives funds from which input.
Input: Shuffle \((O_\pi(1),...,O_\pi(n))\), escrow addresses \(E_1,...,E_n\), partial signatures \((r_1,\langle s'_1 \rangle,\langle k^{-1}_1 \rangle),...,(r_n,\langle s'_n \rangle,\langle k^{-1}_n \rangle)\)

Output: Refund transactions \(E_1^\nu \rightarrow O_\pi(1),...,E_n^\nu \rightarrow O_\pi(n)\)

1. Mixing peers agree on a schedule \(t_1 \leq t_2 \leq ... \leq t_n \leq t_{in} + \omega\).

2. At time \(t_j\), each \(M_i\) computes:
   
   \[ e \leftarrow \text{Hash}(E_j^\nu \rightarrow O_\pi(j)) \]
   \[ (s_j)_i = e \cdot (k^{-1}_j)_i \oplus (s'_j)_i \]
   \[ s_j \leftarrow \text{Recombine}((s_j)_i) \]

3. \(M_i\) announces the refund \(R_i : E_j^\nu \rightarrow O_\pi(j)\) signed by \((r_j, s_j)\) to the Bitcoin network at \(t_i\).

Protocol 3.6 Details of the Transaction protocol: Mixing peers agree on a random schedule to announce transactions to thwart timing correlation attacks. Transactions are constructed on the fly and signed using our modified threshold ECDSA scheme based on [IAIE03].

address. While this does not allow \(M_m\) to link output addresses to users, it undermines the randomness of the final shuffle and violates our correctness requirement (Section 3.2.3). Such misbehavior cannot even be detected by the other mixing peers in the design of [RMSK14] while the protocol of [BS06, CGF10] prevents it at the costs of a complete second encryption layer. We specifically introduce Step 2 in our shuffle protocol to prevent this attack: Note that the permutation \(\pi_{rand}\) applied last is random, if the seed \(C\) is random. A single honest user \(U_j\) can ensure the randomness of \(C\) by choosing her output address \(O_j\) and thus the checksums \(C_j^i\) at random.

We thus fix the vulnerability of the shuffle protocol of [RMSK14] while maintaining superior performance compared to [BS06, CGF10].

With the output addresses decrypted and shuffled, we can now proceed to repay users with the funds held in escrow at \(E_1, ..., E_n\).

### 3.3.7 The Transaction Phase

In the transaction phase, the mixing peers transfer the escrowed funds back to the shuffled output addresses of the users. The details are given in Protocol 3.6. Reusing the PRNG seeded during SHUFFLE, the mixing peers first agree on a random schedule for announcing mixing transactions to the Bitcoin network (Step 1). Distributing transactions randomly over a sufficiently long mixing window thwarts timing correlation attacks and increases anonymity as we analyze and quantify in Section 3.4.2.

Note that approaches depending on Bitcoin’s group transaction scheme must issue all transaction in one batch at the same point in time. The temporal distribution of transactions in CoinParty becomes possible only through our threshold transaction scheme detailed in the following.

To be accepted by the Bitcoin network, the refunds \(E_j^\nu \rightarrow O_\pi(j)\) must be correctly signed with the private key \(d_j\) corresponding to \(E_j\). Since \(d_j\) is shared across all mixing peers, standard ECDSA cannot be used. Instead, the mixing peers need to collaboratively sign the transaction to redeem the funds held in escrow at \(E_j\).

We use TRETHOLD-ECDHA (Protocol 3.2, Section 3.3.3) to realize such threshold Bitcoin transactions. Note that mixing peers only need to complete those steps
3.3. Decentralized Mixing of Digital Currencies

Input: Shuffles \( S^i \) and \( S^{i+1} \), shared checksums \( (C^i_j) \), \( j = 1 \ldots n \)
Output: Set of disqualified users \( \mathcal{U} \) and disqualified mixing peers \( \mathcal{M} \)

1. Identify malicious mixing peers:
   (a) Obtain \( M_i \)'s secret key \( s_{k_i} \leftarrow \text{RECOMBINE}((s_{k_i})) \) and lift encryption layer \( i \) from \( S^i \).
   
   \[ S^{i+1} = \text{Dec}_{k_i}(S^i) = [O_{r_1(1)}]_{k_{i+1}, k_w}, \ldots, [O_{r_1(n)}]_{k_{i+1}, k_w}. \]
   
   (b) If \( S^{i+1} \neq S^{i+1} \) set \( M \leftarrow M \cup \{M_i\} \).
   
   (c) Resume SHUFFLE with \( M_{i+1} \) on \( S^{i+1} \).

2. Identify malicious users:
   (a) Compute checksums \( C^i_{i+1} = \text{RECOMBINE}((C^i_{i+1})) \). Let \( C_{i+1} = \{C^i_{i+1}, \ldots, C^n_{i+1}\} \)
   
   (b) Compute checksums \( c_{i+1} = \text{Hash}(O_{r_1(1)} \ldots , r_1(n)) \).
   
   (c) Compute \( C_{i+1} \leftarrow C_{i+1} \setminus C_r_{i+1} \). Let \( \mathcal{U} \) the set of users who contributed to \( C_{i+1} \).
   
   (d) Invoke \text{TresholdEcdsa} to issue \( E_j \rightarrow I_j \) for all \( U_j \not\in \mathcal{U} \) and abort the mixing.

Protocol 3.7 Details of the \text{BlameAndRecover} protocol: Mixing peers determine misbehaving mixing peers or users and transfer all funds back to the honest users.

of ECDSA in Step 2 that could not be precomputed during the initialization phase (Section 3.3.4). Compared to [IAIE03], our extensive precomputations considerably speed up signature generation during CoinParty’s online phase. To enforce that the signed refund is not broadcasted to the Bitcoin network before the schedule (Step 3), the recombination (Step 2c) is simply delayed by the mixing peers until time \( t_1 \).

If all users supplied correct encryptions and consistent checksums during their commitment and mixing peers shuffled correctly, the mixing has now successfully finished. If any party misbehaved, the additional fifth phase described next is entered.

3.3.8 The Blame and Recover Phase

The fifth phase, blame-and-recover, is entered whenever the shuffle phase fails and has two tasks: i) to determine which mixing peer and user caused an error and ii) to either fix the error or to transfer the escrowed funds back to all honest users. Protocol 3.7 shows all steps in full detail.

By the design of our shuffle protocol (Protocol 3.5, Section 3.3.6), only two possible sources for errors exist: Either mixing peer \( M_i \) did not correctly decrypt the \( i \)th encryption layer and tampered with the shuffle or a malicious user announced a broken layered encryption or inconsistent verification information, e.g., to mount a protocol-level DoS attack against the mixing operation.

We first check whether a mixing peer misbehaved since we can recover from this without aborting the shuffle (Step 1). The mixing peers recombine \( M_i \)'s private session key \( s_{k_i} \), which was generated by all mixing peers during the initialization phase. Using \( s_{k_i} \), the mixing peers reproduce the decryption step in the shuffle protocol from which they obtain a correct shuffle \( S^{i+1} \) (Step 1a). If \( S^{i+1} \) and \( S^{i+1} \) are not equal under permutation (denoted \( \not\equiv \)), \( M_i \) must have tampered with the shuffle (Step 1b). In this case, \( M_i \) is simply skipped and \( M_{i+1} \) resumes the protocol.
on the correct shuffle $S^{i+1}$. Note that this step does not break unlinkability of the shuffle, since two honest mixes are enough to ensure that the final shuffle is random and secret.

The second case, where $M_i$ decrypted correctly but a user misbehaved, is handled in Step 2. Any such user is identified by recombining the set of individual checksums $C_i^{i+1}$ provided by the users in the initialization phase (Step 2a) and checking them against the set of checksums $C_i^{i+1}$ obtained from hashing the encrypted output addresses in the correct shuffle $S^{i+1}$ (Step 2b). Note that $C_i^{i+1} \cap C_i^{i+1}$ is the set of consistent checksums and that user $U_j$ provided $C_j^{i+1} \in C_i^{i+1}$. Thus, we can identify exactly the set of users $\mathcal{U}$ who provided inconsistent checksums (Step 2c).

Reconstructing individual checksums breaks unlinkability and requires that we gracefully abort the shuffle (Step 2d). To this purpose, mixing peers refund all honest users by invoking \textsc{ThresholdEcdsa} (Protocol 3.2, Section 3.3.3). Note that \textsc{ThresholdEcdsa} is guaranteed to succeed even if up to $\tau < m/3$ mixing peers are actively corrupted as we show in Section 3.4.1. Thus, it is assured that no funds of honest users are stolen or lost.

We propose that mixing peers keep the escrowed funds of the non-honest users in $\mathcal{U}$ as a punishment for trying to sabotage the shuffle by providing incorrect or inconsistent inputs. For larger mixing amounts $\nu$, we argue that this mechanism renders these protocol-level DoS attacks economically infeasible. Interestingly, the same mechanism could be applied to punish mixing peers who tamper with the shuffle (additionally to skipping them): Mixing peers could each transfer a certain amount of bitcoins to a collaboratively controlled address as a financial commitment to the mixing operation. If malicious behavior of a mixing peer $M_i$ is then identified in Step 1b, the other mixing peers retain $M_i$'s commitment as a punishment.

3.4 Discussion of System Properties

In the previous section, we have designed CoinParty, a comprehensive system for mixing Bitcoin and related cryptocurrencies. We have claimed that CoinParty is secure against $\tau < m/3$ malicious mixing peers and any number of malicious users, and we have pointed out improvements and advantages over prior works. In this section, we prove these claims and explain how CoinParty fulfills the requirements for an ideal mixing service stated in Section 3.2.3. We first show that CoinParty performs correct mixings (Section 3.4.1), then discuss the achieved anonymity (Section 3.4.2) and deniability (Section 3.4.3). A thorough performance evaluation proves scalability of CoinParty (Section 3.4.4). We conclude with a discussion of cost-efficiency (Section 3.4.5) as well as applicability and usability of CoinParty (Section 3.4.6).

3.4.1 Correctness

We have to show that honest parties succeed to compute the correct output of all protocol phases even in the presence of corrupted users or mixing peers. Our correctness arguments depend on the security of the underlying cryptographic primitives that are provably secure in the targeted malicious adversary model as summarized
3.4. Discussion of System Properties

in Section 3.3.3. In the following, we show that each protocol phase in CoinParty is correct when all users and at most \( \tau < m/3 \) mixing peers are malicious and may arbitrarily deviate from and interfere with the protocol execution. To present a clear overview, we separate the case of malicious users and malicious mixing peers.

3.4.1.1 Correctness of the Initialization Phase

Initialization (Protocol 3.3, Section 3.3.4) sets up the mixing network and precomputes escrow addresses as well as threshold signatures for future mixing operations.

Malicious users. Users do not participate in Initialization (Protocol 3.3, Section 3.3.4). Therefore, it is irrelevant to the correctness of this phase whether and how many users are corrupted.

Malicious mixing peers. In the first part of Initialization, the mixing peers set up session keys. We need to make sure that i) each mixing peer \( M_i \) receives a well-formed key pair \((sk_i, K_i)\) and ii) mixing peers hold consistent shares \(\langle sk_i \rangle\) of the private key. Security of the EcDkg protocol guarantees that generated key pairs \((\langle sk_i \rangle, K_i)\) are correct [GJKR07]. Furthermore, the security of Recombine guarantees that \( M_i \) correctly obtains \( sk_i \) from the shares \(\langle sk_i \rangle\), received from the other mixing peers \( M_j, j \neq i \), even if \( \tau < m/3 \) of them are malicious and send forged or incorrect shares. In the second part of Initialization, the mixing peers generate escrow addresses. Again, the security of the EcDkg protocol guarantees that mixing peers succeed to generate valid ECDSA key pairs. Transforming an ECDSA public key into Bitcoin an address is only a local operation. Thus, each honest mixing peer obtains a correct Bitcoin address. The correctness of the third part of Initialization, the precomputation of the partial signatures, is ensured due to the security of the primitives EcDkg, Add, and Multiply.

Since all used primitives are secure against at least \( \tau < m/3 \) malicious mixing peers and users do not participate, we conclude that the whole Initialization protocol is secure against \( \tau < m/3 \) malicious mixing peers and any number of malicious users.

3.4.1.2 Correctness of the Commitment Phase

During Commitment (Protocol 3.4, Section 3.3.5), a user connects to the mixing network, receives an escrow address, and commits her funds.

Malicious users. In the commitment phase, users provide their encrypted output address as well as corresponding verification information and commit the required funds to the escrow addresses. We first note that, at any point and without any consequences, the mixing peers can stop interacting with a user who fails to provide this information or fails to cooperate otherwise. Furthermore, none of the information provided by the user is actually used in the commitment phase; thus, it is irrelevant to the correctness of this phase. With these observations, the correctness of Commitment boils down to the correctness of the user’s financial commitment \( I_j \rightarrow E_j \) to the assigned escrow address. The correctness of this transaction is ensured by Bitcoin, i.e., completely independent of CoinParty. Furthermore, it follows that collaboration of malicious users does not create new attack vectors. We conclude that the Commitment protocol is secure against any number of malicious users.
3. Decentralized Mixing of Digital Currencies

Malicious mixing peers. An honest user \( U_j \) may have chosen a compromised mixing peer \( M_i \) as entry to the mixing network. The corrupted mix can then try to mount different attacks against \( U_j \). First, the attacker may send \( U_j \) forged identities \((M'_1, \ldots, M'_m)\) and public keys \((K'_1, \ldots, K'_m)\), e.g., those of a mixing network that is under complete control of the attacker. Establishing trust in a set of identities, i.e., those of the true mixing peers \( M_1, \ldots, M_m \), is an orthogonal problem for which well-known solutions exist, ranging from certificate infrastructures [CSF+08] to reputation systems [JHB07] and decentralized trust networks such as PGP [Zim95].

Second, an attacker that has compromised the entry peer \( M_i \) may substitute the chosen escrow address \( E_j \) with another address \( E'_j \) owned by the attacker. However, \( U_j \) validates the mixing parameters and the received escrow address with the other mixing peers. Thus, if the majority of mixing peers is honest, a forged escrow address or set of parameters will be detected by \( U_j \). Finally, a malicious \( M_i \) can prevent any user from joining the mixing by sending a broken session ID \( S_{U_j} \) that is not recognized by the other mixing peers. User \( U_j \) then chooses another mixing peer as entry and will eventually be correctly bootstrapped to the mixing network and the upcoming mixing operation. We conclude that COMMITMENT is secure against \( \tau < m/2 \) malicious mixing peers, i.e., in the presence of an honest majority.

3.4.1.3 Correctness of the Shuffle Phase

We have to show that integrity, randomness, and termination of SHUFFLE (Protocol 3.5, Section 3.5) are guaranteed. Note that unlinkability is proven during the discussion of the anonymity properties of CoinParty in Section 3.4.2.

Malicious users. While users are not directly involved in the protocol, SHUFFLE operates on the input they provided during the commitment phase, i.e., the layered encryption of the output addresses and the secret shares of the corresponding checksums – we only have to consider the case where this information is incorrect or inconsistent.

Clearly, malicious users have no incentive in announcing a wrong or broken output address as this would only result in the loss of their own funds and have no effect at all on other users, the mixing peers, or the mixing operation. However, malicious users can share inconsistent checksums in order to cause the verification of the shuffle stages to fail (Step 1c of Protocol 3.5). In this case, BLAMEANDRECOVER (Protocol 3.7, Section 3.3.8) identifies all misbehaving users, refunds honest users, and aborts the mixing operation. Although we can identify and punish malicious users, it is an open question how to proactively prevent such protocol-level DoS attacks without requiring users to authenticate themselves (which breaks deniability).

Note that prior works [BS06, CGF10, RMSK14] are also vulnerable to protocol-level DoS attacks by a single malicious user. Since these shuffle protocols involve the users in the verification process, they need to abort even when benign users fail inadvertently without any malicious intent. In contrast, our SHUFFLE protocol proposes a verification scheme that does not involve users and is, thus, completely robust against halting users and random inadvertent failures. Our design, furthermore, allows punishing malicious users by retaining their funds, thereby disincentivizing protocol-level DoS attacks economically.
We conclude that our shuffle protocol is fully robust against passively corrupted or halting users while termination against malicious users cannot be guaranteed. The additional BlameAndRecover protocol, however, realizes a graceful abort and allows punishing any malicious behavior among users. The correctness of BlameAndRecover is proved separately in Section 3.4.1.5.

**Malicious mixing peers.** Shuffling in the presence of malicious mixing peers is the most complex case in CoinParty. For the sake of clarity, we split the discussion into three parts: We first show integrity of the shuffle, then prove its randomness, and finally show that malicious mixing peers cannot prevent honest mixing peers from completing the shuffle in a timely manner.

**Integrity.** We define integrity analogous to [BS06]: Either exactly the given output addresses are contained in the final shuffle or the honest mixing peers are informed that some user’s output address has been tampered with. Although our shuffle phase is inspired by [BS06, CGF10, RMSK14], our method of verifying the integrity of the shuffle is fundamentally different and thus requires the following dedicated discussion of correctness.

Any malicious mixing peer $M_i$ can substitute the output addresses in the shuffle with its own output addresses $O'_i$ by announcing $S^{i+1} = ([O'_i], K_{m-1}, ... , [O'_m], K_n)$ since public keys $K_{m-1}, ... , K_n$ are publicly known. However, the honest mixing peers can verify the integrity of the shuffle at each stage $i+1$ using the checksum $C_i$. They will detect the substitution unless the attacker $M_i$ finds $O'_i, ... , O'_m$ such that the checksums $C_i$ and $C'_i$ are equal for all subsequent stages $t \geq i + 1$.

For all but the last mixing peer, it is clearly infeasible to find output addresses such that $C_i$ equals $C'_i$ for all subsequent stages. The reason is that the peers $M_{2..m}$ need to broadcast and thereby commit to the shuffle $S^{i+1}$ before even learning the checksums of the subsequent layers $C'_{i+1}$. Note that all checksums $C_{i+2}, ... , C_{m+1}$ are random if at least one user acted honestly and supplied random individual checksums $C_{i+2}, ... , C_{m+1}$. This is ensured except when all users are compromised — a scenario that clearly does not make sense as the attacker would attack himself only. Hence, $M_i$ cannot predict the checksums and could only try to guess all remaining checksums $C_{i+2}, ... , C_{m+1}$. The probability $p$ of $M_i$ guessing correctly is negligible, e.g., $p \approx 1/2^{256(m-i)}$ for a standard 256 bit cryptographic hash function.

The last mixing peer $M_m$ however, removes the last layer of encryptions and learns the output addresses in clear. $M_m$ can thus derive the checksum $C_{m+1}$ before announcing the shuffle $S^{m+1}$. To steal the mixed funds, $M_m$ must thus find suitable output addresses $O'_1, ... , O'_m$ such that $\sum_{j=1}^{m} \text{Hash}(O'_j) = \sum_{j=1}^{m} \text{Hash}(O_{m+1} \cdots \pi_{m+1}(j))$. Since the attacker can efficiently generate a large number of addresses, this corresponds to solving a high density Random Modular Subset Sum (RMSS) problem. Large problem instances of RMSS as involved in CoinParty are, however, not solvable in reasonable time (as explained in Excursus 3.1). Thus, the attack is thwarted simply by limiting the time for Step 1b of Shuffle and skipping the malicious peer $M_m$ on timeout using the corresponding mechanism of BlameAndRecover.

As an alternative approach, we can also completely prevent $M_m$’s whole attack on the protocol level. To this end, we need to introduce random nonces into checksums on the last stage, i.e., the users provide shares of $C'_{m+1} = \text{Hash}(O_j | n_j)$ during the commitment phase. They also need to provide the random nonces $n_j$ in order to
Decentralized Mixing of Digital Currencies

Excursus: The Random Modular Subset Sum Problem

Finding a sequence \( O_1, \ldots, O_n \) requires solving an RMSS problem instance with modulus \( M = 2^{\text{len}(	ext{Hash})} \), random elements \( \{a_1, \ldots, a_N\} \in \{0, M\} \), and target sum \( C_{m+1} \). \( \delta = N/\log(M) \) is called the density of the problem and high density RMSS instances are very likely to have a solution. The attacker can generate such a high density instance of the problem by precomputing a large number of addresses \( O_i \) and corresponding hashes \( a_i = \text{Hash}(O_i) \in [0, M) \).

The best known algorithm for solving high density instances has runtime \( M^{O(1/\log(N))} \) [Lyu05]. Clearly, the runtime decreases if \( N \) grows, i.e., if we generate a larger pool of hashes to select the \( n \) addresses \( O_i \) from. Contrarily, the runtime increases in \( M \) (the size of the hash function's domain).

We make a practical estimate which sizes of the RMSS problem are solvable in reasonable time. Concretely, we assume our checksum \( C_{m+1} \) is computed from 256 bit hashes, i.e., \( M = 2^{256} \). Then, generating \( 10^{12} \) hashes amounts to 32 TB of storage and we assume it is not feasible to generate and store more than \( N \approx 10^{12} \) such hashes locally for efficient access. Thus, upper bounding \( N \) by \( 10^{12} \) yields a lower bound on the runtime. Assuming there is a solution at all, it can then be found in \( (2^{128})^{1/\log(10^{12})} = 2^{128/12} \approx 2^{42} \) operations which is already quite challenging practically. Using 1536 bit hashes, we estimate a runtime of \( 2^{128} \) operations which is comparable to the complexity of breaking keys for long term security for the Advanced Encryption Standard (AES). Note that using 1536 bit or longer hashes as checksums does not significantly increase the processing and communication overheads of our Shuffle protocol.

Excursus 3.1 An excursus into the RMSS problem, explaining why it is not efficiently solvable in the context of CoinParty for adequate parameter choices.

be able to verify this last shuffle stage. Since the nonces \( n_j \) link in- and output addresses, nonces need to be encrypted and shuffled as well. However, by shuffling and decrypting the nonces only after mixing peer \( M_m \) has announced the shuffle \( S^{m+1} \), \( M_m \) cannot predict the checksum \( C_{m+1} \) anymore and hence cannot mount the RMSS attack, while it is still possible for the honest mixing peers to verify integrity of \( S^{m+1} \). Note that this comes at the cost of a second complete shuffle (of the nonces \( n_j \)). Thus, we would practically prefer the first defense, i.e., imposing a time limit.

Randomness. Due to the lexicographical sorting in Step 2a of Shuffle (verifiable by all mixing peers), the randomness of the final shuffle depends only on the randomness of the final permutation \( \pi_{\text{rand}} \) drawn from the PrNG. The final permutation \( \pi_{\text{rand}} \) is thus random if the seed \( C \) is random which is ensured through the use of a cryptographic hash function (we depend on their pre-image resistance property) for all individual checksums \( C_j \) computed by each user \( U_i \) for each stage \( t \).

Termination. Any malicious mixing peer \( M_i \) may refuse to decrypt the corresponding encryption layer \( i \) or announce an incorrect shuffle \( S^{i+1} \) to stall or halt our shuffle protocol. We recover from this case by skipping the malicious \( M_i \) by entering BlameAndRecover after a timeout.
We conclude that no malicious mixing peer is able to violate the integrity of the shuffle, predict or bias its randomness, or prevent honest mixing peers from executing it correctly. The security of this phase against malicious mixes is only limited by the security of RECOMBINE and the security of the blame-and-recover phase. Both are secure against \( \tau < m/3 \) malicious mixing peers.

### 3.4.1.4 Correctness of the Transaction Phase

TRANSACTION (Protocol 3.6, Section 3.3.7) completes the precomputed partial signatures on the refunds and announces them to the Bitcoin network according to a random schedule to thwart timing attacks. It is crucial that this protocol succeeds to issue these refunds even in the presence of compromised peers since the escrowed funds could be lost otherwise.

**Malicious users.** Users are not directly involved in TRANSACTION and their only input to this protocol are their output addresses. Providing an incorrect output address is not an attack that we need to consider since this harms only the user who announces it but no one else. Therefore, TRANSACTION is secure against any number of malicious users.

**Malicious mixing peers.** Security of TRANSACTION against malicious mixing peers is easy to show. We first note that all honest mixing peers locally draw the same random schedule from the PRNG as they have initialized it on the same seed \( C \) during SHUFFLE. Second, completing the partial signatures precomputed during INITIALIZATION is secure because ADD and RECOMBINE are secure against \( \tau < m/3 \) malicious mixing peers (Section 3.3.3). Finally, the random schedule for announcing transactions is adhered to since honest peers refuse to sign refunds \( E_i \rightarrow O_{\pi(j)} \) before the scheduled release time \( t_j \). Thus, no malicious mixing peer can announce the refund transaction to the Bitcoin network prematurely, e.g., to reduce the anonymity of the mixing, and a single honest mix is enough to ensure that the refund is actually broadcasted when scheduled. We conclude that transaction phase is secure against \( \tau < m/3 \) malicious mixing peers and any number of malicious users.

### 3.4.1.5 Correctness of the Blame-and-Recover Phase

BLAMEANDRECOVER (Protocol 3.7, Section 3.3.8) is invoked whenever the verification of a shuffle stage \( S^{i+1} \) fails. We have already shown in Section 3.3.8 that either the cheating mixing peer or at least one misbehaving user is identified. It remains to show that we can then either skip the mixing peer and continue the shuffle or that we can gracefully abort and refund honest users. Both cases are easy to show.

**Malicious users.** When the shuffle is aborted due to inconsistent input from a malicious user, the mixing peers invoke TRANSACTION to refund honest users. As we have already showed in the previous section, TRANSACTION succeeds in the presence of any number of malicious users and \( \tau < m/3 \) malicious mixing peers.

**Malicious mixing peers.** In the first case, all mixing peers know the last correct shuffle \( S^i \) and can correctly recover \( M_i \)'s private key \( sk_i \) due to the security of SHARE and RECOMBINE. The honest mixing peers can then decrypt layer \( i \) and obtain a
correct shuffle $S^i$ to resume SHUFFLE just as if $M_i$ had behaved honestly. For $\tau < m/3$ and $m \geq 3$, at least two honest mixing peers are not skipped which is enough to ensure unlinkability as explained in Section 3.4.2.1.

We conclude that BlameAndRecover does not undermine the correctness of SHUFFLE and is secure against $\tau < m/3$ malicious mixing peers and any number of malicious users.

### 3.4.2 Anonymity

In the previous section, we showed that honest mixing peers succeed to correctly execute the mixing even in the presence of maliciously corrupted mixing peers or users. In this section, we show that CoinParty achieves the equally important anonymity requirement stated in Section 3.2.3. We first show unlinkability, i.e., that no passive observer of the blockchain, no participating user, and not even a subset of the mixing peers can link a shuffled output address to its owner. We then quantify the level of anonymity achieved through our unlinkable mixing. To this end, we analyze the actual Bitcoin blockchain and show how to maximize anonymity sets.

#### 3.4.2.1 Unlinkability

CoinParty ensures the unlinkability between output addresses and their owners in the shuffle phase. The basic proof for unlinkability is similar to the one presented in [BS06]. However, we need to show that BlameAndRecover does not break unlinkability for correct protocols runs (incorrect shuffles are aborted).

The basic argument for unlinkability between the output addresses and their owners is that by using a semantically secure and length-regular encryption scheme $Enc$, the ciphertexts $[O_{j=1..m}]_{K;K}$ are indistinguishable at each shuffle stage $i$. More precisely, for any shuffle stage $i$ executed by an honest mixing peer $M_i$ (who receives $S^i$, decrypts and secretly shuffles it, then broadcasts $S^{i+1}$), a corrupted mix $M_k$ or outside attacker cannot decide given $S^i$ and $S^{i+1}$ how the $n$ ciphertexts $[O_{j=1..m}]_{K;K}$ correspond to the decryption of the $n$ ciphertexts $[O_{j=1..m}]_{K;K}$. Thus, the attacker cannot link ciphertexts from one shuffle stage $S^i$ to those in the next stage $S^{i+1}$. Stated differently, the attacker cannot observe the permutation $\pi_i$ applied by the honest mixing peer $M_i$ on stage $i$.

We have showed that a single honest mixing peer $M_i$ already ensures unlinkability against any other mix $M_k$, or any outsider. In order to achieve complete unlinkability, i.e., also against $M_i$ itself, only one other honest mixing peer $M_j$ is required: $M_i$ secretly applies a random permutation $\pi_i$, which ensures unlinkability against $M_j$ (and all other mixing peers). Analogously, $M_j$ applies $\pi_j$ to ensure unlinkability against $M_i$ (and all other mixing peers). We hence need to show that at least two honest mixing peers participate in the mixing.

The use of a decryption mixnets principally ensures the participation of all mixing peers because each mixing peer $M_i$ must decrypt the corresponding layer $i$ before the others can proceed. However, as a measure of robustness, we introduced a mechanism that allows honest peers to reconstruct decryption keys of misbehaving mix peers in
order to skip them. Importantly, this mechanism cannot be misused by corrupted mixing peers to skip the shuffle stage of all honest peers in order to break unlinkability since the security guarantees of RECOMBINE prevent any set of $\tau < \frac{m}{3}$ malicious mixes to reconstruct decryption keys of honest mixes. Given our limit of $\tau < \frac{m}{3}$ malicious mixes, furthermore, participation of at least $m - \tau > m - \frac{m}{3} \geq 2$ honest mixes is ensured for all mixnets of size $m \geq 3$ peers.\footnote{A mixnet of size $m = 1$ is equivalent to a classical centralized mix which, logically, cannot achieve unlinkability against itself. Mixnets of size $m = 2$ are not possible in our approach as Shamir’s secret sharing requires $m \geq 3$.}

So far, we have showed that the mixnet successfully unlinks output addresses from the users who submitted them. It remains to show that the individual checksums $C_{i=1...m}$ used for verifying the correctness of each shuffle stage $i = 1...m$ cannot be used in any way that breaks the established unlinkability. Here, the key argument is that mixing peers only hold secret shares $\langle C_{i} \rangle$ of these checksums which are never recombined individually during a successful run of SHUFFLE. Instead, only the sum of the checksums $C_i \leftarrow \text{RECOMBINE}(\sum_{j=1}^{n} \langle C_{j} \rangle)$ is reconstructed for the verification of each shuffle stage $i$, which provides no information about the individual checksums $C_{j}$. Importantly, no set of $\tau < \frac{m}{3}$ malicious mixing peers can reconstruct individual checksums on their own due to the security of SHARE and RECOMBINE.

Unfortunately, the latter argument does not hold when BLAME\text{\textsc{And}}\text{\textsc{Recover}} is invoked on stage $i$ but no misbehavior among the mixes is identified. In this case, at least one user has provided incorrect verification information and the misbehaving users are identified by reconstructing all individual checksums $C_{j}$. Since this breaks unlinkability also for the other honest users, we need to abort the mixing and refund honest users without mixing their funds.

We conclude that unlinkability is guaranteed and protected against $\tau < \frac{m}{3}$ malicious mixing peers while already two honest peers are enough to create unlinkability.

### 3.4.2.2 Anonymity Level

So far, we have shown that CoinParty’s shuffle is correct, random, and unlinkable. However, an adversary can always try to guess the mapping between a user and her output address and it remains to analyze the success chances of this attack. The set of addresses among which the attacker has to guess is the anonymity set and we refer to its size as the achieved anonymity level. A bigger anonymity set leads to a smaller probability of a correct guess and hence more anonymity. In the following, we characterize and quantify the anonymity sets that CoinParty guarantees to honest users against i) any passive observer of the blockchain, ii) other users participating in the mixing, and iii) the mixing peers.

**Passive observers.** Since the Bitcoin blockchain is publicly available, anyone can pose as a passive observer of a CoinParty mixing. Analyzing the anonymity against this kind of attacker, the key insight is that our threshold transactions (Protocol 3.6, Section 3.3.7) are indistinguishable from standard Bitcoin transactions. A passive observer can, however, still try to identify CoinParty’s mixing transactions by i) their correlation in time and ii) the reoccurring mixing value $\nu$. In the following, we characterize the resulting anonymity sets and quantify their size in detail.
Decentralized Mixing of Digital Currencies

Generally, CoinParty produces mixing transaction chains of length two: First, user \( U_j \) commits at least \( \nu \) bitcoins to the escrow address \( E_j \) during the time window \([t_{\text{open}}, t_{\text{in}}]\) in one transaction \( I_j \rightarrow E_j \), where \( t_{\text{open}} \) is the possibly publicly known time when the mixnet started accepting participants. In the transaction phase, which is executed during the time window \([t_{\text{in}}, t_{\text{in}} + \omega]\), the mixing peers transfer \( \nu \) bitcoins from \( E_j \) to another participant’s output address \( O_{\pi(j)} \) in a second transaction \( E_j \rightarrow O_{\pi(j)} \). Note that the mixing amount \( \nu \) is known to the users and we thus assume it is also known to the attacker. A crucial question w.r.t. anonymity is whether this transaction pattern is characteristic enough to distinguish mixing from non-mixing transactions in the blockchain.

Let us first assume that this is the case in order to establish a lower bound for the anonymity level. The attacker can then exactly identify the \( n \) mixing transaction chains of length two produced by CoinParty among all other transactions in the blockchain during the time frame \([t_{\text{open}}, t_{\text{in}} + \omega]\). The attacker however cannot distinguish between the \( n \) transactions belonging to the same mixing operations. Thus, the anonymity level against passive observers is thus at least \( n \), or even \( k \cdot n \) if there are \( k \) contemptuous mixing operations with mixing value \( \nu \) and \( n \) participants each. The latter \( k \cdot n \) bound is due to mix indistinguishability [BNM14], meaning that an observer cannot distinguish in which mixing operation a user participates.

We now show that a passive observer cannot uniquely identify CoinParty’s transaction pattern and that, in consequence, the established lower bound is thus much too pessimistic. As a first step, users can vary the transaction amount in the commitment phase by transferring arbitrary amounts \( \nu' \geq \nu \) to \( E_j \), and the leftovers \( \nu' - \nu \) are repaid by the mixing peers on fresh change addresses. The commitment transaction thereby becomes indistinguishable from all other transactions in \([t_{\text{open}}, t_{\text{in}}]\) with an amount of at least \( \nu \) bitcoins. Unfortunately, we cannot apply the same trick to hide the tell-tale refunds \( R_i : E_i \rightarrow O_{\pi(i)} \) during the transaction phase in the time window \([t_{\text{in}}, t_{\text{in}} + \omega]\): If different amounts \( \nu_1 \neq \nu_2 \neq ... \neq \nu_n \) per user were to be mixed, the mixing peers would need to know which output address \( O_{\pi(i)} \) (i.e., which user) should receive which amount \( \nu_i \). This would require linking users to their output addresses which not only defeats the whole purpose of the mixing but is also not possible as proved in the Section 3.4.2.1.

In consequence, the reoccurring mixing amount \( \nu \) potentially reveals the mixing transactions among the other non-mixing transactions. This reduces the anonymity set to the set of transactions with an output value \( \nu \) during the time window of the mixing. We analyze all transactions in the blockchain between June 2014 and June 2015 in order to determine sensible choices of \( \nu \). We observe that 1 $, 0.1 $, 0.01 $, and 0.001 $ are popular output values, e.g., there are 120,318 transactions for 0.01 $ in June 2014 alone. These values are recommendable mixing values \( \nu \) that promise high anonymity levels.

Even when using these popular values as mixing amounts, releasing all mixing transactions at the same time makes them easily distinguishable from non-mixing transactions based on their strong correlation in time. Thus, a reasonably large mixing window \( \omega \) over which transactions are released to the Bitcoin network is required in order to hide CoinParty’s mixing transactions among normal transactions. We again analyze the blockchain to quantify the increase of the anonymity level depending on the length of the mixing window \( \omega \). To this end, we move a sliding window \( \omega \) of 1 h
3.4. Discussion of System Properties

Figure 3.5 Additional anonymity levels $N_\omega$ (y-axis) for recommended mixing values $\nu$ and different mixing windows $\omega$ (x-axis) during June 2014 to June 2015. The boxes show the 2nd and 3rd quartiles; the whiskers show the range of the data, i.e., minimum and maximum.

Figure 3.5 plots the 2nd and 3rd quartile (boxes) as well as the minimum and maximum (whiskers) increase of anonymity levels for mixing windows $\omega$ of 1h to 48h over all possible points in time during June 2014 to June 2015 for the four recommended mixing values $\nu$ of 0.001, 0.01, 0.1, and 1.0. We observe that increasing the size of the mixing window $\omega$ greatly increases the achieved anonymity level. Already a mixing window of 12h increases the anonymity sets for all considered amounts $\nu$ on average by $N_\omega = 500$ to 1000 transactions compared to a mixing that is carried out instantly. Importantly, these anonymity levels $N_\omega$ are achieved in addition to the base-line of $k \cdot n$ established above, i.e., the total anonymity level achieved against passive observers is $k \cdot n + N_\omega$ for $k \geq 1$ contemptuous mixing operations.

Users. Users actively participate in the mixing and have more information than a passive outsider that may help to de-anonymize fellow mixing participants. In particular, a user $U_j$ can distinguish a mixing transaction from other non-mixing transactions in the blockchain if her input address is mapped to another user’s output address and vice versa. In the following, we explain and quantify how this partial information reduces anonymity.

We first analyze what information a single user learns by participating in the mixing. Through the commitment $C_j: I_j \rightarrow E_j,$ $U_j$ learns that the address $E_j$ is part of the mixing. The user can afterwards observe the refund $R_j: E_j \rightarrow O_{\pi(j)}$ in the blockchain and thereby learn that $O_{\pi(j)}$ is part of the mixing. Vice versa, the user knows her own output address $O_j$ and can hence trace back to $I_{\pi^{-1}(j)}$ over her own refund $R_{\pi^{-1}(j)}: E_{\pi^{-1}(j)} \rightarrow O_j$ and the commitment $C_{\pi^{-1}(j)}: I_{\pi^{-1}(j)} \rightarrow E_{\pi^{-1}(j)}$. With high probability, $I_{\pi^{-1}(j)}$ and $O_{\pi(j)}$ are the addresses of other users since $\pi(j) = j$ has a
Decentralized Mixing of Digital Currencies

Figure 3.6 Attackers chances (y-axis) to de-anonymize a user $U_j$ for different number of users $n$ (lines) depending on the fraction of compromised users $c/n$ (x-axis).

We generalize the case of a single user to an attacker who has compromised $c < n$ of the $n$ users, e.g., via a Sybil attack [Dou02], with $c = 1$ corresponding to the case of a single honest-but-curious or malicious user. The attacker then already knows his own $c$ input and output addresses and can bind other addresses to the mixing by tracing commitment and refund transactions as described above. However, with an increasing number of compromised users the probability of learning new addresses decreases. We model this as a random experiment in which the attacker randomly draws from a set of $n$ elements (the $n$ output addresses) of which $n - c$ are success states (the $n - c$ output addresses the attacker does not know, yet) and $c$ are failure states (his own $c$ output addresses). The involved probabilities are given by a hypergeometric distribution $h(n, n - c, c)$ over a population size $n$ with $n - c$ success states and a sample size of $c$. The expected number of successes is given by $E(h(n, n - c, c)) = c(n - c)/n$. Thus, the attacker is able to learn $c(n - c)/n$ output and $c(n - c)/n$ input addresses of other users.

How can the attacker use this partial information about the mixing to increase his chances of guessing the correct output address $O_j$ of a user $U_j$? When $O_j$ lies in the set of $c(n - c)/n$ addresses tied to the mixing, the attacker has better chances of guessing than when $O_j$ lies in the huge set of $n - c(n - c)/n$ other addresses that are only potentially involved in the mixing. As analyzed in detail in Excursus 3.2, the attacker maximizes his success probability to $p_{\text{success}} = 1/(n - c)$ by always guessing among the tied addresses. The success probability for different numbers of users $n = 10, 50, 100, 200$ is plotted in Figure 3.6. As expected, the attacker’s chances of de-anonymizing a user $U_j$ decrease quickly when more users participate in the mixing.

Perhaps surprisingly, the success probability in our case is exactly the same as that in related works where Bitcoin’s group transaction pattern is used which allows tying all output addresses to the mixing. This shows that the additional anonymity
3.4. Discussion of System Properties

Excursus: De-anonymizing CoinParty Mixings by Guessing

We analyze in detail how an attacker can use partial information about the addresses in the mixing to maximize his chance of guessing the correct output address $O_j$ of a user $U_j$.

Case 1 ($O_j$ has been tied to the mixing): First, we need to determine the probability $P(C_1)$ of this case. Since $O_j$ is contained in a set of $n-c$ addresses from which the attacker learns $c(n-c)/n$ addresses, we have

$$P(C_1) = \frac{c(n-c)/n}{n-c} = \frac{c}{n}$$

However, the attacker still does not know which of the $c(n-c)/n$ tied addresses belongs to user $U_j$. Since the attacker has no further information, she now needs to guess among all candidates. Her success probability is

$$P(G|C_1) = \frac{1}{c(n-c)/n} = \frac{n}{c(n-c)}$$

where $G$ denotes the event of the attacker guessing correctly. We thus obtain

$$P(G \cap C_1) = P(G|C_1) \cdot P(C_1) = \frac{1}{(n-c)}$$

Case 2 ($O_j$ has not been tied to the mixing): First, note that Case 2 is the complement of Case 1 and we can denote Case 2 by $\overline{C_1}$ with

$$P(\overline{C_1}) = 1 - P(C_1) = 1 - \frac{c}{n} = \frac{n-c}{n}$$

Again, what are the attacker’s chances $P(G|\overline{C_1})$ of guessing correctly? Naively, the number of addresses among which the attacker has to guess is $n + N_\omega$. However, the attacker can exclude his own $c$ output addresses and the $c(n-c)/n$ other addresses that have been tied. Thus, the attacker’s chances are

$$P(G|\overline{C_1}) = \frac{1}{n + n-c-c(n-c)/n} = \frac{n}{n N_\omega + (n-c)^2}$$

We thus obtain

$$P(G \cap \overline{C_1}) = P(G|\overline{C_1}) \cdot P(\overline{C_1}) = \frac{n-c}{n N_\omega + (n-c)^2}$$

Importantly, the attacker needs to decide which case to follow, i.e., whether to guess among the $c(n-c)/n$ tied or among the $n + N_\omega + n-c-c(n-c)/n$ untied addresses. Assuming the attacker chooses $C1$ with probability $p_{C_1}$ and $\overline{C1}$ with $1 - p_{C_1}$, then the overall success chances are

$$p_{\text{success}} = p_{C_1} \cdot P(G \cap C_1) + (1 - p_{C_1}) \cdot P(G \cap \overline{C_1})$$

Clearly, $P(G \cap C_1) \geq P(G \cap \overline{C_1})$ for all $N_\omega \geq 0$. With $p_{C_1} = 1$ (i.e., always guessing among the tied addresses) the attacker maximizes success chances to

$$p_{\text{success}} = \frac{1}{n-c}$$

Excursus 3.2 Detailed analysis of the chances to guess a user’s output address.
set of $N_\omega$ against passive observers is not achieved against corrupted users. This insight and the demonstrated attack is interesting since it also applies to commercial centralized mixing services and shows that the anonymity against other users is much smaller than commonly assumed.

In summary, the anonymity of the mixing in CoinParty and related work is significantly reduced if the adversary is able to corrupt a large fraction of the users, which is practical through Sybil attacks. We emphasize that Sybil attacks seem endemic to Bitcoin mixing services: An attacker can use mixing services to generate untraceable Bitcoin addresses which the attacker could then use as input addresses to mount a Sybil attack against another mixing service. Since this attack is passive, honest users or mixing peers cannot recognize it, e.g., in order to punish the attacker as we propose for malicious users that mount DoS attacks. An obvious solution is to make such attacks expensive by charging fees for participation [BOLL14] (cf. Section 3.4.5). This solution is easily applicable in CoinParty but results in a trade-off with our cost efficiency requirement. We are not aware of any related work that solves this problem by design.

Mixing peers. Mixing peers present the third case, different from passive observers and users, since they inevitably learn which output addresses are involved in the mixing operation. However, since mixing peers do not learn which output address belongs to which user, the anonymity level against mixing peers is equal to the number of participants in the mixing $n$ (e.g., the same as for [RMSK14,Yan12] and better than [BNM+14,Max13a]). We are not aware of any related work that achieves a higher anonymity level against the mixing service itself than the number of users $n$ that participate in one mixing operation.

To conclude our anonymity analysis, we remark that CoinParty’s mixing delay $\omega$ is much bigger compared to other distributed mixing services that mix quasi instantaneously in one single group transaction. On the first sight, these increased mixing delays may seem disadvantageous, but the contrary is really the case: The ability of CoinParty to distribute mixing transactions over an arbitrarily long time period is the main reason for the orders of magnitude higher anonymity against passive observers (the anonymity set is increased by $N_\omega$ as depicted in Figure 3.5) in comparison to other decentralized mixing services [Max13a,Rei12,RMSK14,Yan12]. The atomic group transaction pattern used in these works links all mixing transactions together and limits the achieved anonymity set to the number of participating users. Although CoinParty would achieve similar small delays if a window $\omega = 0$ is used, we emphasize that a larger mixing delay must be tolerated if high anonymity levels are desired. This observation is consistent with the results in other application areas of decryption mixnets, e.g., for the anonymous communication networks [LRWW04].

3.4.3 Deniability

We discuss to which extent CoinParty fulfills our third requirement for ideal mixing, plausible deniability, as stated in Section 3.2.3. A user $U_j$ can plausibly deny having participated in a mixing operation if an attacker can present no evidence that binds the user to that particular mixing operation (with high probability). As before, we split our analysis according to the three different attacker types, i) passive observer, ii) other users, and iii) mixing peers.
3.4. Discussion of System Properties

Passive observers. CoinParty does not emit any cryptographic evidence proving to outsiders that a user participated in a mixing, e.g., unlike the approach proposed in [VR15] which leaves irrefutable evidence in a public append-only log, or [Max13a, Ros12, RMSK14], which are based on group transactions that are identifiable as mixing transactions [MO15]. To any passive observer CoinParty’s mixing transactions are, furthermore, indistinguishable from other transactions in the blockchain as explained in Section 3.4.2.2. A user then has means of plausible deniability if there are many more non-mixing than mixing transactions in the Bitcoin network. Our analysis of the blockchain in the previous section shows that there are indeed many non-mixing transactions of the same form as those issued by CoinParty, if a mixing window $\omega$ of sufficient size and popular mixing values $\nu$ are used.

To provide a concrete example, consider a mixing among $n = 100$ users with a value of $\nu = 0.01$ bitcoins over a window of $\omega = 12$h. According to Figure 3.5, there are at least $N_\omega = 1000$ normal transactions that are indistinguishable from CoinParty’s mixing transactions. Without further evidence to the contrary, any user involved in this total of 1100 transactions can plausibly claim that the particular transaction she was involved in is a non-mixing transaction with probability higher than 90%.

Users. As we have argued in Section 3.4.2.2, an adversary who has corrupted a total of $c$ users can tie a random subset of $c$ input addresses to the mixing by tracing back the refund and commitment transactions in the blockchain (and present this as hard evidence). If the attacker can tie user $U_j$’s input address $I_j$ to the mixing, $U_j$ cannot anymore plausibly deny having participated in the mixing as we assume that $I_j$ can be linked to $U_j$ (remember that this is why $U_j$ needs to mix her bitcoins in the first place). For deniability, other than for anonymity, it does not matter whether the attacker is able to tie $O_j$ since the mixing ensures that $O_j$ is unlinkable to $U_j$. Thus, the probability that a user evades the attacker is given by $p_{\text{deny}} = 1 - c/n$, i.e., an honest user’s chances of plausible deniability decrease linearly in the fraction of compromised users.

We emphasize that our shuffle protocol (Section 3.3.6) ensures that the shuffle is random and that no insider or outsider can bias or predict the randomness. This prevents the attacker who has compromised a fraction of the users to mount a targeted attack against the deniability of a particular honest user, i.e., an attacker can only probabilistically attack a particular user and all users are equally likely to evade such an attack with probability $p_{\text{deny}}$.

Mixing peers. CoinParty does not achieve deniability against mixing peers. Mixing peers learn which in- and output addresses participated in the mixing during the shuffle phase and can present user $U_j$’s commitment $I_j \rightarrow E_j$ together with a proof of ownership of $E_j$ as evidence. Deniability against the mixes could potentially be achieved if mixes blindly signed the mixing transactions – blind signatures exist and are already used in related work for other purposes [VR15]. We consider providing deniability against the mixing service itself future work.

3.4.4 Scalability

In this section, we show that CoinParty fulfills the scalability design goal stated in Section 3.2.3. We first explain our experimental setup then present a quantitative evaluation of the runtime and communication overhead of CoinParty in the
initialization, commitment, shuffle, transaction, and blame-and-recover phases. We measure these phases separably since their individual functionality is interesting to other areas and applications beyond our use case, e.g., TRANSACTION can be used to secure Bitcoin wallets [BFK+15, GBF+14].

3.4.4.1 Experimental Setup

We have implemented a prototype of CoinParty in Python 2.7 partly based on code from the VIFF SMC framework [VIF10]. All communication between the mixing peers is handled by the asynchronous communication framework Twisted [LeF02]. All point-to-point connections between mixing peers are secured with TLS and ECDSA is used to additionally sign all protocol messages. All functionality related to the Bitcoin protocol, e.g., generating addresses and transactions, is implemented using the bitcointools4. We also use the pyelliptic5 library for all functionality related to elliptic curve cryptography. In particular, we employ ECIES over the standardized secp256k1 curve [SEC10] for the layered encryption.

We benchmark the selected protocol phases in a local setup in order to quantify the bare processing and communication overheads. A comparison to a cloud setup that accounts for real-world bandwidth constraints and network latency is presented afterwards. In the local setup, each mixing peer is run as a separate process on a single host (16 cores at 2.6 GHz, 32 GB RAM, Ubuntu 14.04 LTS) and communicates with all other peers over the local loopback interface.

We run all considered phases in mixing networks of different sizes $m \in \{4, 7, 11, ..., 31\}$ and for different numbers of participating users $n \in \{10, 50, 100, 200\}$. The choice of the mixnet sizes is due to CoinParty’s security threshold $\tau < m/3$, e.g., a mixnet of size $m = 4$ is secure against one malicious mixing peer, $m = 7$ against two malicious mixes, and so forth. For each phase, we present the total runtime (measured from the start until the last mixing peer finishes) and the communication overhead per mixing peer (since it is equal for all peers). We repeat all experiments in 20 independent runs and provide the mean and standard deviation (denoted by $\sigma$).

3.4.4.2 Evaluation of the Initialization Phase

The initialization protocol has two tasks, i) to set up session keys and ii) to generate escrow addresses and precompute partial signatures. Session keys have to be set up only once for a new mixing network and can then be used for many mixing operations. Only when a mixing peer $M_i$ is falsely accused of malicious behavior, a new session key for $M_i$ has to be created. Since key generation is a one-time effort during the setup of the mixing network, we do not include it in the evaluation of the recurring efforts for setting up a mixing operation.

The overhead for the second task, generating escrow addresses and precomputing partial signatures, scales linearly in the number $n_{max}$ of users that are allowed to the mixing. Thus, we present only the overheads for performing the precomputations

5https://pypi.python.org/pypi/pyelliptic (accessed: 2017-06-20)
for a single user and the total overhead is at most $n_{\text{max}}$ times that overhead (we practically profit from batching). Figure 3.7 plots the total runtime for the mixing network (left) and the communication overhead per mixing peer (right). A network of medium size $m = 13$, e.g., needs 8.83 s ($\sigma = 1.06$ s) and 0.12 MB ($\sigma = 0.00$ MB) to finish precomputations for one user. Note that the host we run our evaluation has only 16 cores. For $m = 19$, at least 3 mixing peers run as hyperthreads which are perceptibly slower and impact the overall runtime. For $m \in \{22, 25, \ldots, 31\}$, this effect is more and more amortized by the overall increasing runtime of \textit{Initialization}. This explains the kink in the curve at $m = 19$.

We emphasize that \textit{Initialization} can be performed at any point in time before the mixing. We can also shift at least two thirds of the overhead, i.e., the overhead for precomputing partial signatures, into the transaction phase, if desired. This makes sense when the mixing window $\omega$ chosen for the transaction phase is longer than the runtime for precomputations. In our most complex setting, i.e., a mixnet of $m = 31$ with $n = 200$ users, the mixing network runs for 2.86 h ($\sigma = 0.16$ h) and each peer communicates 106.50 MB ($\sigma = 0.01$ MB) to finish the initialization phase, of which approximately 1.91 h ($\sigma = 0.11$ h) are spent on precomputing the transaction phase. Thus, even for small mixing windows this overhead could be fully shifted to the transaction phase.

A considerable part of the overhead of \textit{Initialization} is contributed by the \textsc{EcDkg} primitive which is invoked thrice. Note that we could also use pseudo-random secret-sharing to set up shared randomness in the mixing peers. While this incurs one order of magnitude smaller overheads as we show in \cite{ZGH+15}, it also requires a trusted dealer which the fully decentralized \textsc{EcDkg} does not.

### 3.4.4.3 Evaluation of the Commitment Phase

The commitment phase is executed between a user and all of the mixing peers. The overhead on the mixing peers is negligible as it corresponds to answering a simple \texttt{HTTPS GET} request, i.e., the overhead on the mixing peers is equivalent to serving a short static web page. We thus concentrate on the overheads for the user.

The overhead on the user’s side consists in i) contacting the entry peer, ii) contacting all other mix peers to check consistency of mixing parameters, iii) computing the
Decentralized Mixing of Digital Currencies

Layered encryptions and checksums, iv) broadcasting the encryptions and checksums, and, finally, iv) issuing a single Bitcoin transaction. The computation and communication overheads for steps i) and ii) are equivalent to issuing \( m \) short HTTPS GET requests. Step iii) requires \( m \) hash computations, ECIES encryptions, and sharing operations. Step iv) only involves \( m \) short HTTPS POST requests. Finally, Step v) costs only a small constant amount of ECDSA signatures (equal to the number of input addresses from which the user pools and transfers the mixing amount \( \nu \)). We measured the runtime for all steps below 2s for the largest mixnet of \( m = 31 \) peers, which we deem negligible even for mobile users.

3.4.4.4 Evaluation of the Shuffle Phase

Figure 3.8 shows the total runtime (left) and the communication overhead per mixing peer (right) of SHUFFLE for 10 to 200 users and mixnets of size 4 to 31. We observe that both runtime and communication overheads are small even in our largest experiment with \( m = 31 \) mixing peers and \( n = 200 \) users, i.e., the mixnet finishes in only 51.41s (\( \sigma = 0.30 \)s) and transmits 11.89MB (\( \sigma = 0.00 \)MB) per mixing peer.

We also note that the number of participating users has only little impact on the overall runtime, i.e., the difference between shuffling 10 users and 200 users amounts to only 4.84s (\( \sigma = 0.83 \)s) even in our biggest network of 31 mixing peers. The only overhead that depends on the number \( n \) of users in SHUFFLE is due to the required \( n \) ECIES decryptions per shuffle stage. Since elliptic curve cryptography is very efficient, these overheads are very small and are dominated by the overhead for the verification of the \( m \) shuffle stages. Each verification requires one invocation of RECOMBINE which incurs one round of communication between all mixing peers, i.e., overhead quadratic in the size of the mixnet \( m \) but independent of the number of users \( n \).

3.4.4.5 Evaluation of the Transaction Phase

Figure 3.9 plots the total runtime (left) and the communication overhead per mixing peer (right) for performing TRANSACTION for \( n \in \{10, 50, 100, 200\} \) users in mixnets of sizes 4 to 31. In contrast to SHUFFLE, the runtime of TRANSACTION scales
3.4. Discussion of System Properties

Mixnet size

Total runtime [s]

200 users
100 users
50 users
10 users

Traffic per Mix [MB]

Figure 3.9 Overall runtime (left) and communication overheads per peer (right) of the transaction phase for \( n \in \{10, 50, 100, 200\} \) users and mixnets of size \( m \in \{4, 7, 10, \ldots, 31\} \).

linearly in the number of participating users \( n \), e.g., a mixnet of size \( m = 31 \) takes approximately 291.38 s (\( \sigma = 0.93 \) s) and 0.76 MB (\( \sigma = 0.00 \) MB) to refund \( n = 200 \) users. This is due to the RECOMBINE primitive which, other than for SHUFFLE, is invoked \( n \) times during TRANSACTION.

We emphasize again that the refund transactions should be distributed over the whole mixing window to increase anonymity as argued in 3.4.2.2. Thus, the mixnet would have to issue only a few single transactions from time to time. Even in the largest mixnet of size \( m = 31 \), a single transaction costs only 1.46 s (\( \sigma = 0.00 \) s) and 3.78 kB (\( \sigma = 0.00 \) kB) of communication per mixing peer. We conclude that the overheads imposed by the transaction phase are clearly feasible even for large mixnets and many users.

3.4.4.6 Blame-and-Recover Phase

We do not evaluate in detail the blame-and-recover phase, since it has almost the same overhead as the transaction phase. Compared to TRANSACTION, it requires at most two additional invocations of RECOMBINE which are negligible compared to the overhead of TRANSACTION.

3.4.4.7 Cloud Setup

We evaluate each protocol phase again in a cloud setup in order to quantify runtime overheads due to bandwidth constraints and latency in a real-world network scenario. In the cloud setup, each mixing peer is deployed in Microsoft’s Azure Cloud using a separate DS1 instance (1 core at 2.2 GHz, 3.5 GB RAM, Ubuntu 14.04 LTS) for each mixing peer. The virtual machines are distributed over different geographical locations in Western Europe and the Central U.S. The pairwise round trip times between mixing peers are 110 ms between Europe and the U.S. and 5 ms to 10 ms within Europe and the U.S., respectively. Bandwidth was measured at 120 Mbit/s to 500 Mbit/s for inter- and intracontinental communication, respectively. We also limit the setting to the largest mixnet size \( m = 16 \) that could be handled in the local setup without hyperthreading. Since cloud instances also have less processing power than the host used in the local setup, the following results thus present an upper bound for the runtime overheads imposed by communication.
Figure 3.10 compares the runtimes of the Initialization, Shuffle, and Transaction in the local setup (green bars) and the added overheads in the cloud setup (blue bars). We do not evaluate Commitment which only requires mixing peers to serve simple HTTP requests for which we deem the added cloud overheads negligible. As before, we do not evaluate BlameAndRecover since it has almost the same overhead as the transaction phase.

As in the local setup, precomputing escrow addresses and partial signatures during Initialization is the most expensive part with a duration of 19.26 s ($\sigma = 4.20$ s) per participating users in a mixing network of size $m = 16$. This amounts to an added overhead of 5.46 s (39.57 %) compared to the local setup. For $n = 200$ users, the mixnet initialization phase then finishes after 1.07 h ($\sigma = 0.23$ h) of which about 42.81 min ($\sigma = 9.34$ min) are spent on precomputing the transaction phase. Thus, even for our smallest mixing windows we could shift this precomputation from the initialization phase to the transaction phase.

We observe that the cloud setup almost triples the overheads of Shuffle, e.g., shuffling $n = 200$ users takes 6.11 s ($\sigma = 0.33$ s) in the local setup compared to 15.29 s ($\sigma = 0.13$ s) in the cloud setup. This significant increase is due to the efforts for performing reliable broadcasts in each shuffle round. The overhead induced by a reliable broadcast is determined by the maximum latency occurring between any two mixing peers. We exemplarily measured our biggest experiment, i.e., mixing 200 users through 31 mixing peers in the cloud setup and observe a runtime of less than 5 min. This is still much lower than the time the Bitcoin network ultimately needs to validate the final mixing transactions (approximately 60 min at the current block generation rate). Thus, the overhead of the shuffle phase is clearly feasible in the cloud setup, even for large mixing networks and many users.

Finally, the cloud setup adds approximately 37.66 % to the runtime overheads of Transaction, e.g., increasing runtime for $n = 200$ users in a mixnet of size $m = 16$ from 29.01 s ($\sigma = 1.89$ s) to 39.93 s ($\sigma = 0.62$ s). Since transactions should be distributed over the whole mixing window, we consider this overhead to be insignificant. In summary, CoinParty is efficient even in a real-world cloud setup with increased latency and reduced bandwidth between mixing peers.
3.4. Discussion of System Properties

3.4.4.8 Comparison to Related Work

We briefly relate CoinParty’s performance to that of CoinShuffle [RMSK14], which is among the most advanced mixing services and closest to CoinParty in its design. In CoinShuffle, the mixing is executed directly by the users. This corresponds to settings in CoinParty where the number of mixing peers $m$ equals the number of users $n$. According to [RMSK14], a CoinShuffle mixing between $n = 30$ users takes approximately 20s. Running CoinParty in a comparable setting takes approximately 50s for Shuffle and 45s for Transaction. The total of 135s is roughly $7 \times$ the overhead of CoinShuffle. This increased overhead is what we pay for the improved anonymity, plausible deniability, and resilience against protocol-level DoS attacks.

The increased overhead is clearly relativized by the following two observations. First, other than for CoinShuffle, we do not need to scale the number of mixing peers with the number of users. For example, CoinParty mixes $n = 200$ users in a mixnet of size $m = 13$ in approximately 18.12s ($\sigma = 4.40s$) compared to approximately 160s for CoinShuffle. The separation of mixing peers and users thus allows CoinParty to scale much more efficiently to large number of users. Second, we have showed that mixing transactions should be distributed over a mixing window of at least a few hours to prevent their correlation in time. Hence, large fractions of CoinParty’s overheads (i.e., approximately two thirds of the overheads of Initialization and all overheads of Transaction) can be spread over the orders of magnitude larger mixing window.

In summary, we conclude that CoinParty fulfills the stated scalability requirement.

3.4.5 Cost-efficiency

The utilization of mixing services generally involves two kinds of fees, i) transaction fees paid to the Bitcoin miners for including the issued transactions in the blockchain, and ii) mixing fees paid to the mixing services themselves. We briefly discuss both types of fees and show that CoinParty minimizes them.

Transaction fees. A small transaction fee is paid for each Bitcoin transactions to incentivize miners to include the transaction in a newly mined block through which they earn the fee. Historically, transaction fees were voluntary but the exact requirements are subject to change. As of April 2017, a small transaction fee $\mu$ must be paid for any Bitcoin transaction which is proportional to the size of the issued transaction, i.e., $0.0026\, \text{B} \text{per 1kB}$ (cf. Section 3.2.1). When we developed CoinParty in 2014, transaction fees were not yet required for larger mixing amounts $\nu \geq 0.01\, \text{B}$ and transactions of less than 1kB in size. We quickly review how many transaction fees are required in CoinParty back then and today.

CoinParty requires one commitment transaction and issues one refund per user. Each of these individual transactions are approximately 0.25kB in size, hence require a transaction fee of $\mu = 0.00065\, \text{B}$ today and were free of charge when we developed CoinParty. The user can pay the transaction fee for the commitment directly, while the user pays the fees for the refund transaction $R_j: E_j \rightarrow O_{\pi(j)}$ by sending $\nu + \mu$ funds to $E_j$ from which $\nu$ are repaid and $\mu$ is offered as transaction fee to the Bitcoin network. Note that these costs are optimal in the ideal model of mixing we established in Section 3.3.1 since we always need at least two transactions per user (one commitment and one refund).
3. Decentralized Mixing of Digital Currencies

Mixing fees. Anyone, even the users themselves, can set up a CoinParty mixing peer and collaborate with others to build a secure and anonymous mixing network. Thus, CoinParty does not depend on any third party services that could dictate mixing fees. On the other side, as we note in Section 3.4.2.2, it might be reasonable to charge fees to disincentivize Sybil attacks. Mixing fees, however, must be handled with care because they potentially distinguish mixing from non-mixing transactions which decreases anonymity and chances of deniability. Different solutions that are straightforward to adopt in CoinParty such as randomized all-or-nothing fees have been discussed in [BNM+14].

3.4.6 Applicability & Usability

CoinParty extends the standard Bitcoin protocol in two ways. First, CoinParty implements a distributed generation of Bitcoin addresses used for escrow. Apart from the shared private key, escrow addresses are exactly the same as normal Bitcoin addresses. A user can thus commit funds to them using any Bitcoin client. Second, CoinParty introduces threshold transactions to redeem funds from the shared-control escrow addresses. Threshold transactions are generated differently than normal transactions but are otherwise exactly the same. Thus, CoinParty is fully compatible with Bitcoin which we validated in Bitcoin’s test network.

CoinParty is also applicable in different settings. Though we separated users and mixing peers throughout this work, this separation is only logical: CoinParty can be run by a set of dedicated mixing peers or just as well by the users themselves.

In order to provide high usability, each mixing peer in CoinParty runs a web server that allows users to connect to the mixnet via their browser. We also implement the complete commitment phase to run in the user’s browser, so that the user only needs a Bitcoin client or web-based wallet to issue the commitment – no additional non-standard software is required. Furthermore, we require only one short information exchange with each user, i.e., after providing the required information the user never needs to be contacted again. Participating in a CoinParty mixing operation thus requires minimal effort and technical knowledge and is possible even for non-experienced users.

3.5 Conclusion and Future Work

We motivated the problem field of financial privacy which becomes increasingly relevant due to the continuing digitalization of financial services. A promising solution are decentralized cryptographic currencies which aim to provide anonymity properties equivalent to those of conventional cash. However, multiple works show that users can be de-anonymized, e.g., in the blockchain of the widely adopted cryptocurrency Bitcoin [AKR+13,RH11,RS13,MPJ+13,SMZ14]. A prominent line of research thus proposes mixing services [BNM+14,Max13a,RMSK14] to reestablish financial privacy. However, our in-depth analysis of these approaches revealed serious limitations regarding anonymity, deniability, and scalability.
3.5. Conclusion and Future Work

As starting point for our own contribution, we recognized the potential of SMC to provide stronger security and privacy guarantees for mixing services. We then presented CoinParty, a novel decentralized mixing service based on SMC that improves significantly over the related work by combining the advantages of centralized and decentralized mixing services in a single system. CoinParty achieves this by emulating a centralized mix through an SMC protocol that is carried out by dedicated mixing peers. At the core of our secure mixing protocol is a new verifiable oblivious shuffle scheme based on decryption mixnets [RMSK14, CGF10] as well as an efficient threshold ECDSA scheme [IAIE03].

The detailed qualitative and quantitative discussion of our approach clearly shows that the application of SMC has distinct advantages in the context of mixing (as summarized in the beginning of this chapter in Table 3.1): First, the security guarantees of SMC allow us to mix users’ funds in many individual transactions where related work depends on group transactions to achieve security. We showed by an analysis of the actual Bitcoin blockchain that this allows CoinParty to improve upon related work in terms of anonymity and deniability against insiders and outsiders. Second, SMC allows us to logically and physically separate users and mixes, which enables CoinParty to scale to a much larger number of users than related work. Furthermore, through this separation and our new shuffle verification scheme, we achieve for the first time in a distributed mixing service resilience against random failures and deliberate protocol-level DoS attacks of malicious users and mixing peers. Third, building on standard SMC techniques, CoinParty, notably, does not depend on mechanisms specific to Bitcoin, such as its group transaction pattern used in previous proposals. Thus, our contributions do not only extend directly to other cryptocurrencies which use the same ECDSA primitive, e.g., Litecoin and Mastercoin, they are also applicable beyond our immediate use cases, e.g., for securing digital wallets [BFK+15, GBF+14].

We emphasize that modeling CoinParty as an SMC problem and realizing it on top of standard SMC techniques allows us to directly profit from current and future work in this active research field. This makes it much easier to extend, adapt, and enhance CoinParty compared to proposals that build on Bitcoin-specific functionality. It is especially interesting future work to investigate CoinParty under weaker security assumptions: For example, the availability of non-equivocation mechanisms leads to SMC protocols that only require an honest majority [BBCK14]. These techniques directly raise CoinParty’s collusion resistance threshold to $\tau < m/2$ malicious peers. CoinParty could even be secured against a dishonest majority of up to $\tau \leq n - 1$ compromised peers in the slightly weaker covert adversary model [DKL+13].

Our ideas and improvements have been acknowledged, adopted, and further improved by subsequent works that we briefly summarize in the following. The PathShuffle system [MSRK] adopts our idea of using threshold signatures and applies it to mixing in the credit-network Ripple. With DiceMix [RMSK16], the authors improve on the DoS resilience and scalability issues we identified in their previous CoinShuffle scheme. SecureCoin [Ibr17] is a direct combination of CoinParty and CoinShuffle [RMSK14] that adopts our idea of using threshold signatures for mixing to increase anonymity and provide deniability. Mann and Loebenberger [ML15] adapt MacKenzie’s [MR01] two-party DSA signature scheme to elliptic curves to provide two-factor authentication of Bitcoin wallets – a similar system is straightforward to
realize based on CoinParty’s concept of escrow addresses. Bonneau et al. [BFK+15] significantly extend on this idea and propose a novel threshold ECDSA signature scheme that allows realizing Bitcoin wallets with arbitrary access structures. The increased flexibility, however, comes at the price of significantly increased processing and communication overheads. Their proposal represents an interesting generalization of CoinParty’s escrow mechanism for use cases other than mixing – in distributed mixing services, where all mixing peers have the same privileges, such arbitrary access structures are neither required nor do they increase security.

Before we conclude this chapter, we would like to draw the attention to an important point that remained unaddressed so far. CoinParty, as any other technologies providing anonymity such as TOR, can be used for good (e.g., to support dissidents and whistle blowers) and evil (e.g., to launder money and finance criminal activities). Current research seems to largely neglect this issue and focus on providing maximum privacy in digital currencies [CLR+17], although a similar political discussion is currently fought out over backdoors into cryptographic algorithms and breaking encryption. We argue that perhaps the most important direction of future work is thus to investigate practical trade-offs that allow balancing privacy for the individuals with the capability to revoke privacy in case of abuse (which is obviously not limited to the concrete use case targeted in this chapter). Valid scenarios are easy to imagine, e.g., strong suspicion of criminal activities, imminent danger to human life, or drug trafficking. In this light, CoinParty’s security threshold of $\tau < m/3$ is not a limitation but a feature as it allows a quorum of $\tau + 1$ mixing peers to collectively de-anonymize the shuffle and trace abuse – if all mixing peers collaborate they can even de-anonymize only single users such that privacy of all honest users is retained. We view CoinParty as a first step into this important research direction but acknowledge that more work is required on such quorums, e.g., to realize arbitrary access structures (the scheme by Bonneau et al. [BFK+15] is a promising approach, though developed for an entirely different use case) as well as to render decisions to revoke privacy transparent and accountable.

To summarize, we have primarily addressed our first research question (i.e., extending SMC functionality) in this chapter by presenting a compelling use case for SMC and contributing novel SMC functionality for its technical realization. The monetary value involved in our use case, the mixing of digital currencies, required us to consider malicious adversaries that actively cheat, e.g., trying to rob other users of their digital cash, or to block funds. In our design of CoinParty, we thus focused on security in the strongest, the malicious, adversary model with performance being only a secondary design goal. Still, we show that CoinParty scales to realistic problem instances, e.g., is able to mix funds of several hundreds of users in the order of minutes. Since this is more than enough for real-world applications, the use case of mixing cryptocurrencies does not motivate dedicated efficiency improvements (our second research question). In the next chapter, we thus move on to a different problem field, pattern recognition and machine learning, that challenges us with much higher efficiency requirements – to meet these is subject to our second contribution.
In this chapter, we focus on improving the **efficiency** of Secure Two-Party Computation (STC) (a special case of Secure Multiparty Computation as introduced in Chapter 2). Following our use case-driven research methodology, we first select a concrete problem field to motivate and validate our contributions: We identify machine learning as an interesting application area for STC since it has lead to novel digital services that cause widely recognized privacy concerns [EFG+09, PRRS13] but has high performance requirements that challenge state of the art STC protocols (Section 4.1). After a brief background on machine learning (Section 4.2), we analyze the core problems, requirements, and challenges with regards to STC (Section 4.3).

In the first main part of this chapter (Section 4.4), we present SHIELD [ZVHW14, ZMR+17, Met17], our framework for privacy-preserving classification and pattern recognition for the typical scenario where the classification model and input data are held privately by two separate parties, e.g., as in biometric identification [BCP13] or computational genomics [DCFGT12]. We develop accurate STC protocols for Hyperplane classifiers, Artificial Neural Networks (ANNs), Naive Bayes, and the Forward and Viterbi computation on Hidden Markov Models (HMMs). We thoroughly evaluate each design regarding accuracy and performance and show that our approaches outperform the state of the art. To show the applicability of our techniques, we demonstrate different real-world use cases, i.e., privacy-preserving spam filtering [Met17], bioinformatics services [ZMR+17], and indoor localization [ZVHW14].

In these contributions, our main optimization goal is protocol runtime. However, during the development of our use cases, we observe that more and more mobile application scenarios emerge in which other cost factors become important. The significant communication overheads of STC likely overtax mobile users, especially when they connect over cellular networks instead of fast LANs. Focusing on our third research question (**customizability**), we propose two strategies for tailoring STC to mobile application scenarios (Section 4.5): optimizing bandwidth of STC [ZHH+15] and securely outsourcing protocol execution to the cloud [ZMR+17].
4. Privacy-preserving Classification and Pattern Recognition

4.1 Motivation

"Data is the oil of the 21st century economy" is a claim readily made by mainstream media and researchers [MC13]. It compares the role of data in the 21st century digital revolution to the role of oil in the industrial revolution of the 18th and 19th century. Indeed, what is commonly referred to as Big Data technologies, i.e., advances in artificial intelligence based on machine learning coupled to the technical capabilities to collect, store, and process massive amounts of data has arguably transformed businesses, research, and our everyday lives. A particularly widely established manifestation of this development are digital services offering classification and pattern recognition. The list of examples is seemingly endless: Speech and handwriting recognition, biometric identification, localization, spam recognition, stock market predictions, medical diagnosis, and personalized medicine are only a few examples.

These services are commonly deployed in the cloud with access conveniently integrated into desktop and mobile applications, e.g., speech, fingerprint, and face recognition. Users are then required to send their data to the service provider who predicts class labels using proprietary models and returns just the personalized result. This practice neglects that users’ input data is in many cases highly sensitive and must be protected. For example, speech recognition [PRSR11, PR13, PRRS13] not only reveals to the service provider what a user searches for, the recorded speech samples also allow creating voice profiles that could be misused to impersonate users. In a similar vein, handwriting recognition systems [CT02, BLM06, GSB07] could automatically generate a user’s signature from previously collected samples. As a final example, the operator of a medical diagnostics systems [SG11, SG13, WGH12] learns what diseases a user suffers from, which presents valuable information to all kinds of insurance companies. Due to these security and privacy concerns, many users are hesitant to adopt these services [XG09, ZGMW14, Lyo14].

An obvious solution to these problems would be to provide the statistical model to the user who executes the required computations locally on her device. The models used in pattern recognition and classification, however, are expensive to train and the accuracy of the obtained model is what sets the service provider apart from its competitors. To remain competitive, the service provider must hence protect the model just as any other intellectual property. For example, Apple’s Siri, arguably, was the first wide spread speech recognition service also because it was much more accurate than earlier services – handing out its statistical models would have immediately enabled competitors to offer the same service quality on their smartphones. Data protection legislation presents a second reason why handing out models to clients is not a viable solution. For example, consider a genetic disease testing service whose classification models have been trained over confidential patient information. Though the model abstracts from the patient records, residual risk remains that the learned model still leaks information about the conditions of individual patients [EE11]. Thus, sharing the model with the users might be unlawful, e.g., according to the U.S. Health Insurance Portability and Accountability Act of 1996.

We conclude that pattern recognition and classification services face a conflict of business interests, regulatory issues, and privacy concerns. This prompts the questions whether these services can be designed such that neither party learns the other party’s sensitive inputs. We identify STC as a promising solution approach: STC
allows both parties to execute the desired computation under strong security and privacy guarantees, i.e., the service provider’s model is kept secret and the user is guaranteed that nobody learns her inputs or the classification result. Designing secure classification and pattern recognition protocols that are sufficiently efficient and accurate for real-world use cases presents a major challenge.

4.2 Background on Machine Learning

We provide a brief background on machine learning and motivate a representative choice of established approaches to classification and pattern recognition. Our main contribution in this chapter is the design, implementation, and evaluation of STC protocols for these algorithms that outperform and extend upon the state of the art.

In literature, the usage of the terms pattern recognition, machine learning, and classification is ambiguous and overlapping. In this thesis, by classification we denote the basic task of assigning a categorical label to an unlabeled data record. With pattern recognition, we refer to more complex tasks such as labeling each element of an entire time-series. In this context, machine learning is the basic approach to solving these tasks, i.e., algorithms that allow computers to “learn” from a set of training data through statistical inference in order to later automatically classify data or recognize patterns based on the trained statistical models.

4.2.1 Classification

Classification is the task of predicting for an unlabeled data record $d$ a class label $c \in C = \{c_1, ..., c_k\}$. A practical way to approach this problem is through machine learning as illustrated in Figure 4.1. In a first step, a statistical model $M$ is trained on the feature vectors $X = (\vec{x}^1, ..., \vec{x}^m)$, $\vec{x}^i \in \mathbb{R}^n$ extracted from a labeled dataset $D$. This is referred to as supervised learning as opposed to unsupervised learning where the records in the training set are unlabeled, e.g., as in clustering. Using the learned model $M$, the classification algorithm $C : \mathbb{R}^n \rightarrow C$ predicts a class $c = C(M, \vec{x}) \in C$ for a new unlabeled record $d$ based on the feature vector $\vec{x}$ extracted from $d$ in this chapter, we consider the problem of computing $C$ securely to address scenarios where $M$ and $\vec{x}$ are sensitive and held by two mutually distrusting parties.
We put these definitions into context by a short example of detecting rare illnesses. Human doctors face difficulties in diagnosing such illnesses due to their diverse clinical picture and infrequent occurrence. To support them, classifiers are trained on historic patient records and medical studies (the dataset $D$). Note that we can either train many classifiers where each recognizes only one specific disease (i.e., two classes $C = \{\text{influenza, no influenza}\}$) or just a single model which recognizes all considered diseases (i.e., multiple classes $C = \{\text{healthy, fever, influenza, lung-infection, ...}\}$).

Given the electronic health record (the unlabeled record $d$) of a new patient with unexplained symptoms, the system preprocesses and extracts relevant parts (the feature vector $\vec{x}$) and predicts whether the patient has a certain disease or not (i.e., the class $c = C(M, \vec{x})$). In 2016, IBM has demonstrated the viability of this approach by diagnosing a rare form of leukemia that baffled medical doctors [Ng16].

There are of course many different approaches to training models and using them for classification. In the following, we briefly review the three approaches, Hyperplanes, ANNs, and Naive Bayes, for which we present secure protocols later in this chapter. We choose these three for their universality (e.g., perceptron, least squares, Fischer linear discriminant are all based on hyperplanes [Bis06]), their topicality (e.g., recent breakthroughs in artificial intelligence are largely based on ANNs [LBH15]), and their prevalence (e.g., Naive Bayes is a widely used baseline method [Bis06, BPTG15]). Since our goal is to facilitate privacy-preserving classification and pattern recognition services (as motivated above), we assume that the required models have already been trained by the service provider. In the following, we thus focus on the definition of these models and their classification rules.

### 4.2.1.1 Hyperplane classifiers

Hyperplane classifiers [Bis06] achieve classification through a linear combination of the features of a data item $d$, i.e., the classification rule is given by

$$C_{\text{Hyper}}(M, \vec{x}) = f\left(\sum_{i=1}^{n} w_i \cdot x_i\right) = f(\vec{w} \cdot \vec{x})$$

(4.1)

where $\vec{w}$ are trained weights and the function $f(\cdot)$ maps the inner product into two classes, e.g., often using the sign function. In literature, an additional bias $b$ is often subtracted from the dot product which we model by expanding $\vec{w}$ with $w_{n+1} = -b$ and $\vec{x}$ with $x_{n+1} = 1$. The classification model is thus given by $M_{\text{Hyper}} = (\vec{w}, f)$.

We can visualize Hyperplane classifiers by interpreting $\vec{w}$ as the normal vector of a hyperplane $\vec{w} \cdot \vec{x} - b = 0$. This hyperplane splits the $n$ dimensional feature space into two parts. Hyperplane classifiers thus require that data is linearly separable into two classes. They can be generalized to non-linearly separable data using the kernel trick [STC04] and to data with multiple classes through a one-versus-all approach, i.e., training a total of $k$ models where model $M_j$ decides whether a given feature vector $\vec{x}$ belongs to class $c_j \in C$ [BPTG15]:

$$C_{\text{Hyper}}(M_1, \ldots, M_k, \vec{x}) = \arg \max_{c_j \in C} (\vec{w}_j \cdot \vec{x})$$

(4.2)

With these definitions, we can model a range of different classifiers that have linear predictor functions [Bis06], such as Support Vector Machines (SVMs), (multinomial) logistic regression, least squares, perceptrons, and Fisher’s linear discriminant.
4.2. Background on Machine Learning

Input Layer

Hidden Layer 1

Hidden Layer 2

Output Layer

Feature vector $\vec{x}$ with three features using an ANN with two hidden layers. The final class is selected as the $\arg\max$ over all outputs.

4.2.1.2 Artificial Neural Networks

ANNs [Bis95, RN95, Coo98] are inspired by biological neural networks such as the human brain. They are composed of many individual artificial neurons that are organized in different layers. During operation, a single neuron typically computes a weighted sum of all its inputs and fires when this excitation level exceeds a certain threshold, i.e., the neuron’s output is zero if the excitation level is below the threshold and computed using an activation function when above it. Generally, data to be classified is fed into the network at the input layer (layer $l = 0$, with one neuron per feature), propagated through all neurons on the hidden layers (layers $1 \leq l < L$, with arbitrary many neurons), and, finally, the classification result can be read from the neurons on the output layer (layer $l = L$, with one neuron per class).

In feed-forward networks each neuron takes inputs from all neurons on the previous layers and passes its output to all neurons on the subsequent layer, i.e., the network forms a directed acyclic graph. In a recurrent network, neurons can be arbitrarily connected and form cycles. In this chapter, we concentrate on feed-forward networks which are much better understood and easier to compute than recurrent networks which can become unstable and oscillate [RN95, pp. 570-571].

Mathematically, we model a feed-forward ANNs as a function that is composed of the activation functions of the individual neurons [RN95, pp. 567-570]. In particular, the $i^{th}$ neuron on layer $l \geq 1$ is modeled by

$$y_l^i = \varphi^l \left( \sum_{j=1}^{m_{l-1}} w_{l,j}^i \cdot y_{l-1}^j \right) = \varphi^l \left( \vec{w}_l^i \cdot \vec{y}_{l-1} \right)$$

(4.3)

where $\vec{w}_l^i = (w_{1,i}^l, \ldots, w_{m_{l-1},i}^l)$ are the synaptic weights between the $i^{th}$ neuron on the $l^{th}$ layer and the neurons on the previous layer $l-1$, $\vec{y}_{l-1}$ the outputs of those neurons, and $\varphi^l(\cdot)$ the activation function shared by all neurons on layer $l$. Commonly used activation functions are the step, sign, and sigmoid functions [RN95, p. 569]. Input neurons get only the $i^{th}$ feature $x_i$ as input and thus the weighted sum and activation function is usually omitted, i.e., $y_0^i = x_i$. Figure 4.2 illustrates this model by the example of an ANN with $L = 3$ layers (not counting the input layer) and $m_1 = 3$, $m_2 = 2$, and $m_3 = 2$ neurons per layers $l = 1, 2,$ and $3$, respectively.
In summary, an ANN is defined by its weight vectors and activation functions, i.e., $M_{\text{ANN}} = (\vec{w}_1, \vec{w}_2, ..., \vec{w}_L, \varphi_1, ..., \varphi_L)$. The classification rule is given by

$$C_{\text{ANN}}(M, \vec{x}) = \text{argmax}_{c \in C} y_j$$

(4.4)

In its simplest form, an ANN consists of a single intermediate neuron that computes a weighted sum of the features, referred to as a single-layer perceptron. The perceptron corresponds almost exactly to the previously introduced Hyperplane classifiers. A single layer perceptron is thus limited to binary classification problems and linearly separable data [RN95, pp. 573-574]. ANNs with more neurons and layers can handle much more complicated data and classification problems.

4.2.1.3 Naive Bayes

A Naive Bayes classifier is a conditional probability model that assigns probabilities $P(C = c_j | X = \vec{x})$ for all classes $c_j \in C$ to all possible feature vectors $\vec{x}$ [RN95, Chap. 14]. Classification is then a simple matter of selecting the most probable class, i.e.,

$$C_{\text{Bayes}}(M, \vec{x}) = \text{argmax}_{c_j \in C} p(c_j | \vec{x})$$

(4.5)

Often, it is infeasible to learn the posteriors $p(c_j | \vec{x})$ directly from the data [AKCS00, APKf00], e.g., for very large or high dimensional feature spaces. In these cases, the Bayes theorem can be applied to compute the posteriors:

$$p(c_j | \vec{x}) = \frac{p(\vec{x} | c_j) \cdot p(c_j)}{p(\vec{x})}$$

(4.6)

The likelihoods $p(\vec{x} | c_j)$, the priors $p(c_j)$, and the evidence $p(\vec{x})$ are learned from the training data $\mathcal{D}$, e.g., by estimating the parameters of an assumed underlying Gaussian or multinomial distribution. Regarding the distribution of the likelihoods, $P(X|C)$, each feature $x_i \in \vec{x}$ (modeled by random variable $X_i$) is assumed to be conditionally independent from each other feature, i.e., $P(X_i, X_j | C) = P(X_i | C) \cdot P(X_j | C)$, which allows computing the posteriors by

$$p(c_j | \vec{x}) = \frac{\prod_{i=1}^{n} p(x_i | c_j) \cdot p(c_j)}{p(\vec{x})}$$

(4.7)

The classification model $M = (P(X_1 | C), ..., P(X_n | C), P(C), P(X))$ is then given by the distribution of the likelihoods, priors, and evidence.

This classifier is called naive because the central assumption of conditional independence actually does not hold for most real datasets. Perhaps surprisingly, Naive Bayes classifiers have been shown to still provide good results in real-world applications [Ris01, Zhu04]. One popular real-world application of Naive Bayes classifiers is spam filtering [AKCS00]. We briefly explain this example since we demonstrate

\[\text{For simplicity, we only explicitly note the random variables } C \text{ (modeling the classes) and } X \text{ (modeling the features) where necessary and use the shorthand } p(c_j | \vec{x}) \text{ whenever possible.}\]
privacy-preserving spam filtering as a use case later. The spam filter examines each word \( w_i \) from an email message \( m = \{ w_1, \ldots, w_n \} \) and computes

\[
    p(\text{Spam}|w_i) = \frac{p(w_i|\text{Spam}) \cdot p(\text{Spam})}{p(w_i)}
\]

where \( p(w_i|\text{Spam}) \) is the probability that \( w_i \) appears in a spam mail, \( p(\text{Spam}) \) is the overall fraction of spam mails, and \( p(w_i) \) is the probability to observe \( w_i \) in the training set. All distributions can be learned by counting the respective events in the training set. Assuming conditional independence, we can combine the individual probabilities to the overall probability that \( m \) is spam:

\[
    p(\text{Spam}|m) = \prod_{i=1}^{n} p(\text{Spam}|w_i) \cdot p(\text{Spam})
\]

\subsubsection{4.2.1.4 Regular versus Cost-sensitive Classification}

The classification rules given in Equations 4.2, 4.4, and 4.5 correspond to regular classification which assumes all types of misclassifications are equally severe and thus minimizes the classification error rate. In contrast, cost-sensitive classification allows assigning different costs to each type of misclassification [Dom99]. In spam filtering, e.g., discarding a benign mail (a false positive) has higher costs than letting a spam mail slip through (a false negative) [AKCS00].

For cost-sensitive classification, decision rules must thus be modified to minimize the expected costs of the classification decision. In the previous three classification algorithms, this can be achieved by interpreting the outputs as class probabilities:

\[
    C_{\text{cost}}(M, \vec{x}) = \arg\min_{c_i \in C} \sum_{c_j \in C} f_{\text{cost}}(i, j) \cdot p(c_j|\vec{x})
\]

where \( f_{\text{cost}}(i, j) \) is the cost of predicting \( c_i \) for a data item with true class \( c_j \) and \( p(c_j|\vec{x}) \) is the normalized output of the classifier interpreted as the probability of \( \vec{x} \) belonging to class \( c_j \). Regular classification thus corresponds to cost sensitive classification with a cost function \( f_{\text{cost}}(i, i) = -1 \) and \( f_{\text{cost}}(i, j) = 1 \) for \( i \neq j \).

\subsubsection{4.2.2 Pattern Recognition with Hidden Markov Models}

The traditional classification task treated so far can be generalized to the sequence labeling problem in which each element of a sequence should be assigned a class. Although we can reduce this problem to a set of independent classification tasks, sequence labeling often involves time-series data where the classification accuracy can be improved by considering also nearby elements or even the entire sequence when assigning classes. Sequence labeling is a typical task in (temporal) pattern recognition with many real-world applications, e.g., part-of-speech tagging, localization and navigation, speech, gesture recognition, or sequence alignments in bioinformatics. A representative approach to sequence labeling and other problems in (temporal) pattern recognition are HMMs. Since applications of HMMs go well beyond classification, we treat them in a different section than the previous classification models.
In general, HMMs are used to model stochastic processes whose internal state and corresponding transitions are hidden and can only be deduced from the output of the process. An HMM is defined by the tuple $\lambda = (S, A, V, \pi)$ (left side of Figure 4.3). The set $S = \{s_1, ..., s_N\}$ are the possible internal states of the HMM with $A \in \mathbb{R}^{N \times N}$ the state transition matrix, i.e., $a_{ji} = p(s_i | s_j)$ is the probability that the HMM moves from state $s_j$ into state $s_i$. The states of the HMM are hidden and cannot be observed directly but only inferred from the emissions the HMM outputs depending on its current state. The alphabet of emissions is defined by $V = \{v_1, ..., v_M\}$ with $B \in \mathbb{R}^{N \times M}$ the emission probability matrix, i.e., $b_i(v_j) := b_{ij} = p(v_j | s_i)$ is the probability that the HMM emits $v_j$ in state $s_i$. Finally, the initial state distribution $\pi \in \mathbb{R}^N$ defines the probabilities $\pi_i = p(s_1)$ that the HMM’s initial state is $s_i$. The output of the HMM (right side of Figure 4.3) is a sequence of observed emission symbols $O = o_1 ... o_T \in V^{1 \times T}$ referred to as an observation sequence (each $o_t$ could be viewed as a separate feature vector $\vec{x}_t$ in the classical classification setting).

We illustrate these definitions by the short example of a WiFi-based indoor localization system. In such a system, the building is separated into discrete locations (i.e., the states $S$ of the HMM) and movement between locations (the state transition probabilities $A$) must respect constraints of human mobility, e.g., no walking through walls. Since the user’s location cannot be observed directly, the localization system periodically measures the received signal strength to the user’s smartphone (the emissions $V$ of the HMM) and estimates the user’s position using a signal propagation model of the building (the emission probability matrix $B$). The localizations begins either at the user’s last known position or at the entrance of the building (according to the initial state distribution $\pi$).

**Forward algorithm:** Two important problems are associated with HMMs. In the first problem, filtering, we ask for the probability $P(O | \lambda)$ that a given HMM $\lambda$ generated a given sequence of observations $O$. Filtering can be efficiently computed using the Forward algorithm detailed in the following:

- **Initialization:** $\alpha_t(i) = \pi_i \cdot b_i(o_t) \quad \forall i = 1 \ldots N$
- **Recursion:** $\alpha_t(i) = \left( \sum_{j=1}^{N} \alpha_{t-1}(j) \cdot a_{ji} \right) \cdot b_i(o_t) \quad \forall t = 2 \ldots T, \forall i = 1 \ldots N$
- **Termination:** $P(O | \lambda) = \sum_{i=1}^{N} \alpha_T(i)$
4.3. Problem Analysis

Viterbi algorithm: In the second problem, decoding, we want to determine the most probable sequence of hidden states \( S^* \) the HMM \( \lambda \) passed through while emitting the observation sequence \( O \). This problem is solved by the Viterbi algorithm:

**Initialization:**
\[
\alpha_1(i) = \pi_i \cdot b_i(o_1) \quad \forall i = 1 \ldots N
\]

**Recursion:**
\[
\alpha_t(i) = \max_{j=1}^{N} \alpha_{t-1}(j) \cdot a_{ji} \cdot b_i(o_t) \quad \forall t = 2 \ldots T, \forall i = 1 \ldots N
\]

**Termination:**
\[
P(O,S^*|\lambda) = \sum_{i=1}^{N} \alpha_T(i)
\]

**Backtracking:**
\[
s_{t-1}^* = \upsilon_t(s_t^*) \quad \forall t = T \ldots 2
\]

Continuing our example of indoor localization, filtering would be used to train the localization model from a set of labeled sample paths, while decoding would be used for the live tracing of a new user’s most probable path \( S^* \) through the building.

Both algorithms compute the desired outcome iteratively using dynamic programming. Note that the probabilities \( \alpha_t(i) \) get progressively smaller which quickly causes underflows and introduces numerical instability in the computation – a known problem [Rab89, DEKM98] which we also face in our use cases of HMMs. Rabiner [Rab89] proposes to normalize the forward variables \( \alpha_t(i) \) after each iteration to deal with this problem. As an alternative, Durbin et al. [DEKM98] propose to compute and store all values in logarithmic space. We refer to probabilities in logspace as scores with \( \log(p(O|\lambda)) \) and \( \log(p(O,S^*|\lambda)) \) the Forward and Viterbi score, respectively.

4.3 Problem Analysis

Having provided a brief background on classification and pattern recognition based on machine learning, we now analyze in detail the problem of securing these approaches using STC. To this end, we first present a concise problem statement, then analyze to what extent related work addresses the identified problems and requirements, and conclude with a short summary of our own contributions.

4.3.1 Problem Statement

As illustrated in Figure 4.4, we consider two parties, a service provider \( S \) who holds a trained model \( M \) (e.g., a classifier or HMM) and a user \( U \) who holds an input \( x \) (e.g., a feature vector or observation sequence). Together, \( U \) and \( S \) want to compute \( F(M,x) \) (e.g., classifying, filtering, or decoding \( x \) under the model \( M \)). Due to privacy concerns, business interests, or regulatory and legal requirements, neither party is willing to share their inputs with the other or any third party as indicated by the trust spheres in Figure 4.4. Indeed, this scenario is ubiquitous in different application areas of classification and pattern recognition such as genetic disease testing [FDH+11], speech recognition [SS07, PRRS13], biometric authentication [PR13, EFG*09, SSW09], and localization [ZVHW14].

In this chapter, we thus pose the question how \( U \) and \( S \) can compute the desired results obliviously, without learning each other’s inputs. Such a solution not only reconciles the evident conflict of business and privacy interests. Remaining oblivious
Figure 4.4 The problem scenario: User (left) and service provider (right) wish to compute a classification or pattern recognition task. Neither party can share its inputs with the other due to privacy concerns or regulatory issues. Our secure classification and pattern recognition protocols allow the two parties to reconcile their privacy conflicts.

to U’s sensitive inputs, the service provider $S$ also does not have to fear the negative consequences of disclosure of customer data in case of attacks [Fra16, Men16], database leaks [Pri17], or seizure by governments [Lad14, Lyo14].

Surveying real-world applications of classification and pattern recognition, we distill requirements and design goals for corresponding secure protocols that allow realizing privacy-preserving variants of these services.

Performance: We find that efficiency is generally of high importance in most real-world applications of classification and pattern recognition algorithms. When applied in user-centric applications, classification performance is usually optimized for low latency, i.e., the time it takes to classify a single data record. In contrast, data mining applications usually classify large batches of data records such that the throughput of a classification algorithm becomes the primary optimization goal. Note that lowering classification latency automatically increases throughput while the opposite is not necessarily true, e.g., when optimizing for a Single-Instruction-Multiple-Data (SIMD) mode of operation.

We concretize these observations by the following examples of the real-world usage of HMM-based pattern recognition: In speech recognition [Rab89], forward computation over a small five-state HMM is used to recognize a single word or utterance – high throughput is required to handle large vocabularies while low latency is equally important in order to timely react to users’ input. In contrast, biosequence alignment requires to search databases of thousands of HMMs with models reaching hundreds of states [pfa15] – high throughput forward and Viterbi computation are required to find matches in reasonable time. For indoor localization [VLW+12], as a final example, the number of states of a single HMM through which we can efficiently compute a Viterbi path directly determines the localization and navigation accuracy – low latency is the main goal so that a user’s position can be timely and continuously updated.

To ensure the applicability of secure classification and pattern recognition in its many diverse use cases, the first requirement is to minimize overheads of secure classification and pattern recognition protocols. We aim for low latency and high throughput but prioritize latency over throughput where necessary.
4.3. Problem Analysis

**Accuracy:** Ideally, secure protocols for classification and pattern recognition should compute identical results to a standard insecure implementation on plaintexts. However, cryptographic protocols operate over integers while classification and pattern recognition algorithms operate over real-valued probabilities and features. Secure protocols to handle non-integers either require heavy quantization [FDH+10, CS10] or introduce significant overheads [ABZS13, DDK+15]. Fortunately, minor inaccuracies may be tolerable in practice, allowing us to balance between performance and accuracy: In speech recognition, we are only interested in the best matching word while the exact probabilities are less important. In sequence alignments, the goal is to separate matching from non-matching models, i.e., we only determine whether probabilities exceed a certain threshold. In localization, the resolution of the state space poses an upper limit on localization accuracy, hence handling a huge number of states with low numerical precision can result in a more accurate positioning than handling a small number of states with high precision in the same time. Since the actual required numerical accuracy depends on the respective use case, we require that accuracy can be traded off against performance. Note that the obvious approach of reducing the size and complexity of the classification model and feature space is usually not viable, since such modifications require expert knowledge and involve expensive retraining of the learned models. Instead, we aim for trade-offs between performance and numerical accuracy that are independent of the machine learning model and inputs.

**Security:** We define the capabilities of the user and the service provider to attack the computation by the semi-honest adversary model. As defined in detail in Section 2.2, the semi-honest attacker correctly follows the protocol but may try to infer additional information from the protocol transcript. The semi-honest model is a reasonable choice for our problem scenario since the user and service provider both have a strong interest in executing the computation correctly. Compared to security in the malicious model, the semi-honest model then allows for much more efficient protocols while still protecting against insiders and outsiders that try to learn the private inputs of the two parties. We further observe that data involved in typical application areas of classification and HMMs, e.g., in bioinformatics or speech, remain sensitive for longer periods of time. To sufficiently protect such data, we thus argue that we must either provide protection in the information theoretic model or, in the case of a computationally bounded adversary, parametrize all involved cryptographic primitives for long-term security, e.g., by choosing adequate key lengths as recommended by NIST [BBB+07].

**Mobile Users and Constrained Environments:** The increasing number of mobile users, e.g., using speech-to-text services, poses additional challenges to secure classification and HMM computation. Due to limitations regarding processing, communication, and energy, mobile users may not be able to carry out computations themselves. Protocols for secure classification and pattern recognition should thus tailor to such constrained deployment and operation scenarios. A popular solution approach in the related literature is the outsourcing of costly computations from the constrained user devices to more capable peers,


100 4. Privacy-preserving Classification and Pattern Recognition

e.g., a computation cloud. To present a real alternative to constrained users,
outsourcing must be both very efficient and uphold all security guarantees.
Since outsourcing is currently the most promising approach, we pay special
attention to the support for outsourcing in our analysis of related work as well
as in the design of our own protocols (without precluding other approaches).

We deem especially performance and accuracy vital requirements for the applicabil-
ity of secure classification and pattern recognition services.

4.3.2 Related Work

We analyze to which extent prior works address our requirements and goals. First, we
present related work divided into privacy-preserving classification and HMM-based
pattern recognition focusing on the most advanced recent works. Afterwards, we
briefly discuss orthogonal works.

4.3.2.1 Secure Classification

Vaidya and Clifton [VC04] present a secure Naive Bayes classifier based on
Homomorphic Encryption (HE). The practicality of their scheme is questionable
since the authors present no evaluation of its performance or accuracy. The lat-
ter point is especially critical since the proposed protocol freely composes previous
building blocks that have not been proven accurate either.

Yu et al. [YJV06] present HE-based protocols for training and classification with
SVMs. Their approach is restricted by design to binary features and the perfor-
manse is not evaluated. Also, their protocols are not built upon established secure
computation techniques; a dedicated security proof is necessary but missing.

Graepel et al. [GLN12] present secure training and classification protocols for
linear means and Fisher’s linear discriminant classifiers based on Somewhat Homo-

morphic Encryption (SWHE). The resulting classification protocol requires only two-
rounds of communication which is attractive in networks with high latency. They
assume a different problem setting where a single user holds both the training and
testing data and aims to outsource training and classification steps to an untrusted
cloud. While this potentially makes for a promising approach to our requirement
of outsourcing computations, the author’s evaluation shows that the majority of
overheads are actually due to encrypting data on the user’s side and cannot be
outsourced. A defensive performance comparison also indicates that our approach
outperforms theirs even in networks with high latencies.

Bost et al. [BPTG15] present different designs for secure classifiers, i.e., Hy-
perplane classifiers, multinomial Naive Bayes classifiers, as well as decision trees.
The authors identify reoccurring subtasks and propose modular designs based on a
condensed set of building blocks. To realize the corresponding protocols securely,
Bost et al. employ three different HE schemes, i.e., Paillier [Pai99,DJ01], Brakerski-
Gentry-Vaikuntanathan [BGV12], and Goldwasser-Micali [GM82].
The problem scenario and goals of Bost et al. are very similar to ours. Indeed, our secure Naive Bayes and Hyperplane classifiers are notable improvements of their work regarding performance as we evaluate in detail in Section 4.4.8. Additionally, we present further general purpose classifier designs such as an alternative Bayes classifier with support for continuous feature distributions and fully fledged feed-forward ANNs with arbitrary activation functions.

Dowlin et al. [DGBL+16] realize secure ANNs based on an improved variant of the fully homomorphic Brakerski-Gentry-Vaikuntanathan cryptosystem [BLLN13]. The use of a Fully Homomorphic Encryption (FHE) scheme has two main advantages. First, the resulting protocol runs in only two rounds, i.e., the user encrypts and sends its inputs to the service provider who executes the classification in the encrypted domain and sends back only the result. A two-round protocol of course copes much better with network latency than secure protocols that require frequent interaction between protocol parties such as those based on the Goldreich-Micali-Wigderson (GMW) protocol or arithmetic sharings (cf. Section 2.3). Second, recent FHE schemes are designed for SIMD type of operations which significantly increases the throughput in scenarios where a single user has to perform a large batch of classifications. However, building the protocol design on FHE also comes with disadvantages. If a user wants to classify only a few inputs, the proposed approach incurs significant overheads which also cannot be outsourced by the user. Furthermore, the computation of non-polynomial functions, e.g., as required for the common maximum or sigmoid neural activation functions, is very expensive under FHE. The authors argue that such activation functions can partly be approximated through polynomial ones while leaving others out of their design which limits the generality of their approach with respect to more complex ANNs and other classifiers.

Our own approach builds on additive arithmetic sharings which allows us to trade throughput for much lower classification latency and thus represents a completely different point in the design space of secure ANN classifiers compared to the approach by Dowlin et al. [DGBL+16]. Additionally, we support more complex neural activation functions through a novel building block for secure function approximation.

**Summary.** Practical secure variants of the Naive Bayes, Hyperplane, and ANN classifiers already exist, e.g., the approaches by Bost et al. [BPTG15] and Dowlin et al. [DGBL+16]. As we will show, these prior works can be improved in three directions: First, performance is still the limiting factor to the real-world applicability of secure classifiers – we improve the performance of the Naive Bayes and Hyperplane classifiers proposed in [BPTG15]. Second, the proposed classifiers are still tailored to rather specific settings – we show how to generalize their functionality by supporting more complex activation function for ANNs or continuous probability distributions. Finally, prior works do not consider mobile users with limited resources – we achieve secure classification also for constrained users by designing all our secure classifier schemes to support outsourcing and reducing bandwidth consumption.

### 4.3.2.2 Secure Pattern Recognition

Smaragdis et al. [SS07] were first to consider privacy-preserving HMM computation in the context of speech recognition. Their approach is based on HE throughout. HE causes high performance overheads, especially for long-term security levels.
Without an evaluation of performance overheads, it thus remains unclear whether their approach is practical. This is aggravated by the fact that many of their secure protocols, e.g., the inner product, require plaintext knowledge of the inputs, which prevents secure outsourcing. A further concern is the numerical stability of the results, especially during Forward computation: The Forward algorithm involves computation over probabilities, i.e., real numbers, while the cryptographic primitives of their approach operate over the integers. The authors neither explain how they represent probabilities as integers nor do they quantify the involved errors and how they propagate. Finally, missing discussions of security and the use of insecure primitives render the overall security of their approach doubtful.

Pathak et al. [PRSR11, PRRS13] adapt and improve the techniques from [SS07] to keyword recognition and speaker identification. Their approach, as well, makes heavy use of HE which is expensive and scales poorly to long-term security levels as confirmed by their evaluation. Similar to Smaragdis et al. [SS07], outsourcing is not possible since some subprotocols require plaintext knowledge. Interestingly, the authors show that a fixed-point representation of the involved probabilities indeed provides reasonable accuracy on a small five-state HMMs when normalizing forward probabilities after each recursion step. However, it remains questionable whether this approach achieves sufficient accuracy on larger HMMs and observation sequences. It is also left unclear whether and how the security issues of [SS07] were fixed.

Polat et al. [PDRO10] present a different approach based on additive blindings which promises high efficiency. However, their approach uses additive blinding over probabilities as if they were integers from a finite field which raises serious concerns about the numerical accuracy of the computation. The authors provide only a limited evaluation of the performance of their subprotocols on random inputs and do not quantify the numerical accuracy. As in the previous approaches and for the same reasons, outsourcing is not possible. Furthermore, they rely on a Trusted Third Party (TTP) to generate correlated randomness for computing scalar products. If the TTP colludes with either party, the other party’s privacy is lost. The approach by Polat et al. is thus not secure in the semi-honest adversary model.

Franz et al. [FDH+10] propose a framework for secure computations on non-integer values in logarithmic representation which they apply to secure bioinformatics services [FDH+11]. Their approach is first to provide reasonable accuracy for computations on real-world HMMs and observation sequences. It uses lookup tables to compute the critical logsum operation that causes problems in the approaches by Smaragdis et al. [SS07] and Pathak et al. [PRSR11, PRRS13]. The size of the lookup tables constitutes a trade-off between performance (smaller tables) and accuracy (larger tables). However, the size of the lookup tables grows exponentially in the bitlength of inputs which renders their solution rather inefficient when very accurate results are required. As this approach frequently relies on HE primitives, it scales poorly to long-term security levels and cannot be fully outsourced.

Aliasgari et al. [ABZS13, AB13], Kamm et al. [KW15], and Demmler et al. [DDK+15] propose provably secure floating point primitives in the multi- and two-party setting that can be fully outsourced. The proposed primitives could be used to implement the Forward and Viterbi algorithm in a secure manner but none of these works presents a concrete implementation. This leaves unclear whether standard IEEE 754 floating point numbers achieve sufficient accuracy or whether additional
measures are required to avoid underflows of the very small probabilities involved in Forward computation [Rab89]. Furthermore, the performance comparison of the proposed primitives presented in [DDK+15] indicates significant overheads.

Aliasgari et al. [ABB16] compute the Viterbi algorithm in the two- and multi-party setting with security against semi-honest and malicious adversaries. The protocols in the two-party setting are based on HE while threshold secret sharing is used in the multi-party setting; security against malicious adversaries is achieved through zero-knowledge proofs. To achieve reasonable accuracy, the authors build on and extend the secure floating point primitives presented in [ABZS13]. The evaluation of their two-party setup indicates significant overheads in the order of hours even for very small HMMs. The authors justify these runtimes arguing that the HMM itself must be hidden even from the service provider and stored only in encrypted form. Notably, this corresponds to the outsourcing scenario we consider in Section 4.5.2 and can also be covered in our approach at almost no additional costs.

Summary. None of the analyzed approaches provides a fully satisfying solution to the problem of securely computing the Forward and Viterbi algorithm. Notably, most prior works depend on HE primitives that scale poorly to long-term security levels. We conclude that the efficient, accurate, and secure computation of the HMM Forward and Viterbi algorithm is still an open and important problem.

4.3.2.3 Orthogonal Work

Privacy preserving learning on partitioned data. Different works consider how classifiers are trained in a privacy-preserving manner (i.e., without revealing training data to others) over horizontally or vertically partitioned datasets. In vertical partitioned data, each party knows only a subset of the features, while in horizontally partitioned data, each party holds only a subset of the records. Wright et al. [WY04] present an approach for secure Bayesian learning on vertically partitioned data using homomorphic encryption. Vaidya et al. [VKC08] present secure approaches for the training of secure Naive Bayes classifiers on both vertically partitioned and horizontally partitioned data using HE and arithmetic sharings. Kantarcioglu et al. [KVC03] present a secure learning algorithm for Naive Bayes classifiers based on the GMW protocol [GMW87] over horizontally partitioned data. Similarly, Bonawitz et al. [BIK+16] show how to train ANNs on horizontally partitioned data. Lindell et al. [LP06] present a protocol to securely learn ID3 decision trees using Garbled Circuits (GCs).

The common assumption of these proposals is that the learned model is not privacy sensitive and can simply be handed to the users who may then execute the classification algorithm locally on the plaintext model and data record. In contrast, we assume that the learned classifier itself is sensitive and requires protection, e.g., due to privacy concerns, business interests, or legal requirements.

Specific use cases. Finally, multiple other works on secure classification and pattern recognition are highly specialized to single use cases. Bos et al. [BLN14] present a predictor for cardiovascular diseases based on a logistic regression model. However, the authors assume that the classifier is public knowledge and only the user’s input must be hidden during classification. The proposed algorithms thus do not apply
to our setting where nothing must be learned about the model and the input other
than what is implied in the computed result. Barni et al. [BFK+09, BFL+11b] show
how to securely evaluate linear branching programs and neural networks specialized
towards the classification of electrocardiograms. Further, there have been multiple
proposals for face recognition systems implementing either the Viola-Jones [VJ01]
or Eigenfaces [TP91] algorithms securely using HE [EFG+09], GCs [SSW09], or a
combination of Oblivious Transfer (OT) and arithmetic sharings [AB06].
These approaches are highly specialized to their specific use cases and thus do not
generalize easily to different classification tasks. In contrast, we aim to implement
efficient general purpose classifiers that apply to a wide range of classification tasks.

4.3.3 Our Contributions

We propose SHIELD [ZVHW14, ZIH+15, ZMR+17], our framework of protocol de-
signs for privacy-preserving classification and pattern recognition in the STC setting.
SHIELD allows two mutually distrusting parties to efficiently and accurately com-
pute classification and pattern recognition tasks while remaining oblivious to the
private inputs. The core idea of our approach is to combine state of the art
STC techniques, i.e., additive secret sharing, GCs, and OT, with our own novel building
blocks in an efficient manner. The following are our detailed contributions:

Improved and novel building blocks: We propose a suite of secure protocols for
efficiently and accurately computing over probabilities which improve on prior
approaches in terms of performance and numerical stability. They are designed
for modular composition with respect to accuracy and security such that secure
protocols for classification and pattern recognition can be built on top.

Secure designs for classification and pattern recognition: Based on and extend-
ing our secure primitives, we develop Hyperplane, ANN, and NaiveBayes,
secure versions of the corresponding established classification algorithms. We
evaluate our classifiers on multiple widely used datasets, showing that they
are accurate and outperform the state of the art by up to one order of mag-
nitude. We then propose Forward and Viterbi, two efficient protocols for
HMM-based pattern recognition. These two algorithms proved very challenging
in prior works; no approach achieved both reasonable accuracy and practical
performance. Our evaluation shows that our approach has tunable accuracy
and is faster than prior proposals by up to two orders of magnitude.

Real-world use cases: To validate the applicability of the developed secure classifi-
cation and pattern recognition algorithms, we implement three real-world use
cases, i.e., privacy-preserving spam filtering, bioinformatics, and localization
services. In all use cases, the results show that our approach achieves high
accuracy and practical overheads even on large real-world problem instances.

Optimizations for mobile deployments: Despite our significant performance im-
provements, especially the remaining communication overheads may still over-
tax mobile users. We thus present bandwidth-optimized designs for comput-
ing the max and arg max functions which are universally used in classification
and pattern recognition. Our proposed building blocks trade local processing for a significant reduction of communication overheads by 18% up to 98% compared to related work, which even results in overall reduced runtimes in bandwidth-constrained networks. As second, fully compatible and even more radical strategy, we show how all protocols in SHIELD can be outsourced securely and efficiently to an untrusted computation cloud. Our evaluation shows that the involved overheads are feasible even in very constrained environments.

### 4.4 Secure Classification Framework and Designs

We have motivated the need for secure protocols for classification and pattern recognition that allow users and services to keep their sensitive inputs and valuable models private. Our analysis of related work reveals different limitations regarding accuracy, performance, or functionality that motivate us to reconsider this important problem scenario. Especially, the design of accurate and secure protocols for Forward and Viterbi computation on HMMs appears very challenging, as we found no approach in related work (Section 4.3.2) that provides a satisfiable solution.

Our approach to overcome the identified shortcomings in related work is threefold: First, to achieve reasonable accuracy, we represent all involved probabilities with fixed-point precision (and in logspace, where adequate) which allows us to avoid the overheads due to fully-fledged secure floating point primitives \([ABZS13, DDK^{+}15]\).

Second, based on the ideas proposed in \([PRT15]\), we replace the critical logsum operation that caused prohibitive inaccuracies and high overheads in related works \([SS07, PRSR11, PRRS13, FDH^{+}11]\) with a piece-wise linear approximation that we compute securely through an efficient combination of GCs and additive sharings.

Third, we replace the expensive HE primitives of related work with additive secret sharing and OT which is more efficient, scales much better to long-term security levels, and, as we will see later, supports the secure outsourcing of computations from constrained devices to the cloud with almost negligible overheads.

To provide a clear and easy-to-follow presentation, we successively present secure protocols for increasingly complex classifiers and introduce our improved or novel building blocks as we go – culminating in accurate, efficient, and secure protocols.

---

**Figure 4.5** Overview of the structure of this section and the dependencies of our secure classification and pattern recognition protocols (gray boxes) on the different secure building blocks (white boxes); building blocks are presented when they are first required.
for the challenging HMM Forward and Viterbi algorithms. An overview of all proposed classifiers and the required building blocks is given in Figure 4.5 (we proceed top-down from left to right): All our approaches involve computations over real numbers (mostly probabilities), and we start by explaining how we represent them with fixed-point precision as integers such that they can be handled by the cryptographic techniques underlying almost all generic STC protocols (Section 4.4.1). We then construct a Hyperplane classifier (Section 4.4.2) and introduce the required building blocks for the secure computation of scalar products, max, and arg max. Following, we extend our Hyperplane classifier to a full-fledged ANN (Section 4.4.3), introducing a mechanism for securely approximating non-linear functions. Switching to a different flavor of classification algorithms, we present two versions of Naive Bayes classifiers, one for discrete and one for continuous features (Section 4.4.4) as well as corresponding primitives for securely evaluating probability mass and density functions. Finally, we explain in detail our secure protocols for Forward (Section 4.4.5) and Viterbi computation (Section 4.4.6) that combine the gained insights and most of the introduced building blocks, extending it by a final primitive for securely computing logsums.

Having presented the secure classifier designs, we show that all our protocols are secure in the chosen semi-honest adversary model (Section 4.4.7), evaluate their accuracy and performance in detail (Section 4.4.8), and demonstrate their applicability in real-world use cases (Section 4.4.9).

The entirety of our secure building blocks and classifiers makes up SHIELD, our STC framework for privacy-preserving classification and pattern recognition which balances accuracy and performance, allows outsourcing, and scales nicely to long-term security levels. With SHIELD we improve on the performance of prior works, solve the open problem of secure HMM-based pattern recognition, and present techniques that are interesting beyond our application context.

### 4.4.1 Representation of Real Numbers

All classification algorithms considered in this chapter use floating point numbers in practice, e.g., Hyperplane and ANN classifiers involve real-valued weight vectors and activation functions while Naive Bayes, Forward, and Viterbi compute over real-valued probabilities. However, cryptographic primitives typically operate over the discrete algebraic structures, e.g., additive sharings and the conversions to GCs are defined over $\mathbb{Z}_l$, HE over $\mathbb{F}_n$, and OT exchanges $l$ bit integers (cf. Section 2.3).

One solution to this problem are secure floating point primitives, i.e., higher-level protocols that implement floating point arithmetic on top of the established lower-level integer-based STC protocols. Existing proposals for secure floating point arithmetic are based on HE [AB13], additive sharings among three fixed parties [KW15], or combine GCs with commercial logic synthesis software to minimize circuits [DDK+15]. These approaches always provide single or double floating point precision and do not allow trading numerical accuracy against performance. We will see that single precision floats already provide higher numerical accuracy than actually necessary for the simpler classifiers, e.g., Naive Bayes and Hyperplanes. In these cases, the high performance overheads of secure floating point primitives are not justified.
Bost et al. [BPTG15] take the opposite approach and propose to multiply all non-integers \( v_i \) by a constant \( K \) such that \( K \cdot v_i \in \mathbb{N} \). While this is a loss-free conversion, it incurs a significant blow-up as the scaled values reach hundreds of bits in length. Bost et al. argue that all values still fit into the plaintext space of the utilized HE schemes; the popular Paillier scheme, e.g., has a plaintext spaces of 1024 bit to 3072 bit and can easily accommodate the scaled values. However, these values do not fit into the plaintext spaces of our additive sharing techniques, which is typically \( \mathbb{Z}_2^l \) with \( l \in \{32, 64\} \) so that all arithmetic can be handled natively by modern compilers and processors without the need for special big integer libraries. With regard to our main goal of increasing efficiency, we thus dismiss this approach as well.

**Secure Computation with Fixed-point Precision**

For our approach, we choose to represent reals in fixed-point precision as proposed for the multi-party setting in [CS10]. We can think of fixed-point values as rational numbers that are represented by a single integer with a virtual decimal point at a fixed position. Formally, we transform a value \( x \in \mathbb{R} \) to an unsigned integer \( x' \in \mathbb{N}^+ \) by scaling it with a factor \( 2^s \), rounding to the nearest integer, and mapping the result into \( \mathbb{Z}_2^l \), i.e.,

\[
\forall 2i(x, l, s) = \lfloor 2^s \cdot x \rfloor \mod 2^l
\]  

(4.11)

Note that our encoding \( \forall 2i \) (float-to-integer) and arithmetic in \( \mathbb{Z}_2^l \) preserves arithmetic over signed integers when we interpret the result in \( (2^{l-1}, 2^l - 1] \), i.e.,

\[
\forall 2i(x', l, s) = \begin{cases} 
\frac{(x' - 2^s)}{2^s}, & \text{if } x' > 2^l - 1 \\
\frac{x'}{2^s}, & \text{otherwise}
\end{cases}
\]  

(4.12)

After transforming all inputs (e.g., model parameters and feature vectors in the context of classification) using \( \forall 2i \), any intermediate values and results are also expressed in this fixed-point representation. Hence, we must ensure that no value exceeds the bitlength \( l \) to avoid incorrectness due to over- and underflows. The sum of two scaled values is again an integer that is scaled by the same factor and its bitlength increases by at most one which is uncritical. However, multiplication leads to an accumulation of the scaling factor \( 2^s \), i.e., the product of two numbers in our representation is scaled by \( 2^{2s} \), and the bitlengths of the two factors add up which would quickly overflow the maximum bitlength \( l \) and cause errors when another value scaled only by \( 2^s \) is added. To prevent this, we need to scale the product down by the factor \( 2^s \) before any subsequent additions or multiplications can be performed on it.

On plaintext values, rescaling would be a simple matter of division and rounding. In our approach, all values are, however, additively shared in \( \mathbb{Z}_2^l \) which prevents straightforward division and we also cannot recombine them for rescaling without violating our security requirements. Instead, we propose the efficient and secure **Rescale** (Protocol 4.1) based on the proposal due to Catrina and Saxena [CS10, Section 3.1]. Their proposal, however, uses secret sharing in the multi-party setting, so that we need to adapt it to our approach (additive shares in the two-party setting).
Input: Additive sharing \( \langle x \rangle \) of \( x \in \mathbb{Z}_2^l \)

Output: Additive sharing \( \langle x' \rangle \) of \( x' = \lfloor x/2^s \rfloor \in \mathbb{Z}_2^{l-1} \)

\( U \Rightarrow S: \langle x \rangle_U = \langle x \rangle_U \oplus r \) with \( r \in \mathbb{Z}_2^l \)

\( U: \langle x' \rangle_U = -(r \gg s) \mod 2^{l-s} \)

\( S: x_r = \langle x \rangle_S \oplus \langle x_r \rangle_U \)

\( \langle x' \rangle_S = (x_r \gg s) \mod 2^{l-s} \)

Protocol 4.1: Our secure Rescale protocol adapted to additive sharings in the two-party setting from [CS10]. By recombining a blinded \( x_r \), \( U \) and \( S \) are able to securely compute additive shares of \( \lfloor x/2^s \rfloor \) in \( \mathbb{Z}_2^{l-s} \).

In our adapted protocol all operations are performed in \( \mathbb{Z} \), i.e., without modular arithmetic. Initially, \( U \) and \( S \) hold shares \( \langle x \rangle_U \) and \( \langle x \rangle_S \) of an intermediate value \( x \in \mathbb{Z}_2^l \) which is scaled by the factor \( 2^s \). To scale down to \( 2^s \), \( U \) first blinds her individual share \( \langle x \rangle_U \) using a random number \( r \) of length \( l + \kappa \) bit and sends it to \( S \). Then, \( U \) truncates the lower \( s \) bits of \( r \) (a right shift by \( s \) bit scales down by \( 2^s \)), negates the result, and uses it as her share \( x'U \). \( S \) obtains the blinded input \( x_r = \langle x \rangle_S + \langle x \rangle_U \), similarly truncates the lower \( s \) bits of \( x_r \), and uses the result as its share \( \langle x' \rangle_S \). The resulting values \( \langle x' \rangle_U, \langle x' \rangle_S \) share the desired downscaled value \( x' \) in \( \mathbb{Z}_2^{l-s} \). Note that Rescale always rounds down and may thus introduce an error in the least significant bit of the rescaled value. It is, however, more efficient than the alternative exact rounding protocol proposed in [Fra11].

Our introduced fixed-point representation affords a size efficient representation of real numbers that promises increased performance but also introduces numerical errors. We will thus need to carefully evaluate whether our classification algorithms remain sufficiently accurate despite that. Indeed, we find that this simple approach is not sufficient for Forward and Viterbi algorithms as they involve computations over extremely small probabilities that decrease progressively in the size of the HMM and observation sequence. We discuss a solution in the next subsection.

Secure Computation with Fixed-point Precision in Logspace

Aliasgari et al. [AB13, ABB16] argue that full floating point precision is required for computations over HMM. Indeed, the numerical instabilities we encountered when reproducing the approach by Polat et al. [PDRO10] (cf. Section 4.3.2) underline that the dynamic range of the occurring probabilities is indeed too large to compute with fixed-point precision in probability space. However, the considerable computational complexity of Forward and Viterbi computation renders the application of the previously discussed floating point primitives [AB13, ABB16, KW15, DDK+15] very expensive, e.g., Aliasgari et al. [ABB16] report runtimes in the order of hours already for very small HMMs which we consider prohibitive for most real-world applications.

We decide instead to follow the alternative approach by Durbin et al. [DEKM98] and carry out all computations in logspace such that the dynamic range of involved probabilities is reduced to a range that can be handled in our above fixed-point representation. As the results presented in [FDH+11, PRRS13] indicate, this approach (it will require one more building block for computing logsums that we introduce later) achieves sufficiently accurate results for small to medium sized HMMs and ob-
4.4. Secure Classification Framework and Designs

Input: U has feature vector $\vec{x} \in \mathbb{R}^n$
S has $k$ hyperplanes $(M_1 = \vec{w}_1, \ldots, M_k = \vec{w}_k)$ with $\vec{w}_j \in \mathbb{R}^n$

Output: Class $c^* = C_{\text{Hyperplane}}((M_1, \ldots, M_k), \vec{x})$

Initialize shares:
- $U: (x_i)_U = f_{2i}(x_i), (w_{j,i})_U = 0 \quad \forall i = 1 \ldots n, \forall j = 1 \ldots k$
- $S: (x_i)_S = 0, (w_{j,i})_S = f_{2i}(w_{j,i}) \quad \forall i = 1 \ldots n, \forall j = 1 \ldots k$

Compute distance to each hyperplanes:
- $U \leftrightarrow S: (z_j) \leftarrow \text{ScalarProduct}((\vec{x}), (\vec{w}_j)) \quad \forall j = 1 \ldots k$

Determine most probable class:
- $U \leftrightarrow S: (c^*) \leftarrow \text{Argmax}((z_1), \ldots, (z_k))$
- $U \leftrightarrow S: c^* \leftarrow \text{Recombine}(c^*)$

Protocol 4.2 The secure Hyperplane classifier protocol.

4.4.2 Hyperplane Classifier

As a start, we set out to realize the simple yet essential Hyperplane classifier through a secure protocol in a setting where one party holds $k$ models $M_j$ (each consisting of the normal vector $\vec{w}_j$ of a Hyperplane separating the feature space according to class $c_j$) and the other party holds the feature vector $\vec{x}$. To this end, we need to compute Equation 4.2 securely as described in Protocol 4.2. The first step is for user $U$ and service provider $S$ to initialize shares of the normal vectors $\vec{w}_j$ and feature vector $\vec{x}$. Each party uses $f_{2i}$ to initialize shares of its own inputs and sets shares of the other party’s inputs to zero (we denote this as a dummy sharing since there is no interaction between $U$ and $S$ and no values are actually shared). $U$ and $S$ then compute one scalar product for each pair $\vec{w}_j, \vec{x}$ using the secure ScalarProduct protocol (introduced in the following subsection) from which each obtains an additive share of the result. We compute all scalar products in parallel in one round of communication to improve performance. To determine the target class, $U$ and $S$ invoke the secure Argmax protocol (introduced after the next subsection) on the shared scalar products $(z_j)$. The desired result $c^* \in \mathbb{N}$ can be recombined by $U, S$, or both, as desired. Note that $c^*$ is actually the index of the target class $c_* \in C$ and for simplicity we assume from now on w.l.o.g. $C = \{1, \ldots, k\}$ so that we can drop this distinction.
Cost-sensitive classification. It is straightforward to turn Hyperplane into a cost-sensitive classifier (cf. Equation 4.10, Section 4.2.1). To this end, we first compute the shared outputs \( \langle z \rangle \) of the inner product \( z = \langle \vec{x}, \vec{w} \rangle \).

\[
U \leftrightarrow S : \quad \langle z \rangle = \langle \vec{x}, \vec{w} \rangle \quad \forall i = 1...n
\]

Output: Additive sharing of vector \( \langle \vec{x} \rangle \) of length \( n \).

\[
U \leftrightarrow S : \quad \langle \vec{x} \rangle = \sum_{i=1}^{n} \langle x_i \rangle
\]

Output: Additive shares \( \langle \vec{y} \rangle \) of feature vector \( \vec{x} \).

\[
U \leftrightarrow S : \quad \langle \vec{y} \rangle = \text{Rescale}(\langle \vec{x} \rangle)
\]

Protocol 4.3: The secure ScalarProduct protocol based on additive sharings.

Input: Additive sharing of vector \( \langle \vec{x} \rangle = ((x_1),...,(x_n)) \)

Output: Additive sharing of vector \( \langle \vec{y} \rangle = ((y_1),...,(y_n)) \) of feature vector \( \vec{x} \).

\[
\begin{align*}
U & \leftrightarrow S : \quad \langle \vec{x} \rangle \leftarrow \text{GC} \langle x_i \rangle \quad \forall i = 1...n \quad \text{(GC)}
\end{align*}
\]

\[
U \leftrightarrow S : \quad \langle \vec{y} \rangle \leftarrow \text{GC} \langle y_i \rangle \quad \forall i = 1...n
\]

Protocol 4.4: The secure MaxArgmax primitive: \( U \) and \( S \) hold a vector of additive shares and compute the maximum and its index in a GC and return the result as additive shares.

\[
\begin{align*}
\rightarrow & \quad \langle \vec{z} \rangle = \text{MaxArgmax}(\langle \vec{x} \rangle, \langle \vec{y} \rangle)
\end{align*}
\]

4.4.2.1 Secure Scalar Products

Protocol 4.3 securely computes scalar products on additive shares. It is straightforward given addition and multiplication on additive shares (Section 2.3.3.1) and the rescaling protocol introduced in (Protocol 4.1, Section 4.4.1). Both parties multiply the \( n \) pairs of values, add resulting shares locally to compute the sum of the products (which are still scaled by \( 2^s \)), and invoke Rescale on the final result to restore the correct scaling by factor \( 2^s \). To improve efficiency, we batch all messages required for the \( n \) parallel multiplications, resulting in only two rounds of communication.

4.4.2.2 Secure Max and Argmax

The MaxArgmax (Protocol 4.4) protocol securely computes the maximum value and its index in a vector \( \vec{x} \) (max and argmax). At the start, \( U \) and \( S \) both hold additive shares \( \langle \vec{x} \rangle = ((x_1),...,(x_n)) \). They input these shares into \( n \) parallel garbled addition circuits to convert the elements from additive shares \( \langle x_i \rangle \) to garbled...
4.4. Secure Classification Framework and Designs

Input: $U$ has feature vector $\bar{x} \in \mathbb{R}^n$
$S$ has ANN $M = (\bar{w}^1, \bar{w}^2, \ldots, \bar{w}^L, \phi^1, \ldots, \phi^L)$

Output: Class $c^\ast = \text{ANN}(M, \bar{x})$

Initialize shares:
$U: \langle w^l_{i,j} \rangle_U = 0$, $\langle x^i \rangle_U = \text{f}_2(x^i)$ $\forall l = 1 \cdots L, \forall i = 1 \cdots m^l, \forall j = 1 \cdots m^l - 1$

$S: \langle w^l_{i,j} \rangle_S = \text{f}_2(\bar{w}^l_{i,j})$, $\langle x^i \rangle_S = 0$ $\forall l = 1 \cdots L, \forall i = 1 \cdots m^l, \forall j = 1 \cdots m^l - 1$

Feed-forward through all layers $l = 1 \cdots L$:

$U \leftrightarrow S: \langle e^l_i \rangle \leftarrow \text{ScalarProduct}(\langle \bar{y}^{l-1} \rangle, \langle \bar{w}^l_i \rangle)$ $\forall i = 1 \cdots m^l$

$U \leftrightarrow S: \langle y^L_i \rangle \leftarrow \text{FunctionApprox}(\langle e^l_i \rangle, \phi^L)$

Determine most probable class:

$U \leftrightarrow S: \langle c^\ast \rangle \leftarrow \text{Argmax}(\langle y^L_1 \rangle, \ldots, \langle y^L_k \rangle)$

$U \leftrightarrow S: c^\ast \leftarrow \text{Recombine}(\langle c^\ast \rangle)$

Protocol 4.5 The secure ANN classifier protocol.

values $\bar{x}$. On the garbled values, $U$ and $S$ can efficiently select the maximum and its index using a Boolean circuit composed of pairwise comparisons and multiplexers arranged in a tree [KSS09]. As a result, they obtain the garbled maximum $\tilde{x}^\ast$ and its index $\tilde{i}^\ast$ which they convert back to additive shares $\langle x^\ast \rangle$, $\langle i^\ast \rangle$ by computing the OT-based subtraction circuit of [DSZ15].

Note that computing the argmax causes approximately one third of the overheads of MAXARGMAX. If we only need to compute the max (denoted by MAX), we can simply leave out the corresponding part in MAXARGMAX. Note that this does not work nearly as well the other way around, since to compute the argmax we always need to compute the max as well. However, we can save a tiny amount of overheads by not returning the max (for the sake of completeness, we denote this case by ARGMAX but practically employ MAXARGMAX for simplicity).

4.4.3 Artificial Neural Networks

Hyperplane classifiers, as realized in the previous section, allow to model any classifier with a linear predictor function and, in particular, enable us augment a wide range of classifiers (among them the Hyperplane, ANN, and Naive Bayes classifiers presented in this chapter) with custom cost metrics. They are, however, limited to linearly separable data. In this section, we thus consider feed-forward ANNs which are a powerful generalization of simple Hyperplane classifiers and overcome this severe limitation. Since ANNs essentially combine linear classification with non-linear activation functions, we can reuse the building blocks developed for HYPERPLANE.

Protocol 4.5 provides the details of our secure ANN classification protocol. We first initialize all shares using dummy sharing as before for HYPERPLANE. For each layer $l = 1 \cdots L$ and each neuron $i = 1 \cdots m^l$ on layer $l$, we then compute the excitation level $e^l_i$ by the scalar product of the outputs $\langle \bar{y}^{l-1} \rangle$ of the previous layer and the synaptic weights $\bar{w}^l_i$ (Equation 4.3). Since the outputs of the previous layer are either held by $U$ (in the case of layer 0, $\bar{y}^0 = \bar{x}$) or shared additively between $U$ and $S$, and the synaptic weights are held by $S$, we need to employ the SCALARPRODUCT primitive to compute the excitation level securely. In the next steps, we need to evaluate
the activation function $\varphi'$ on the additively shared excitation level $\langle \ell_i \rangle$ to compute additive shares of the output $\langle y_i \rangle$. To this end, we use the FunctionApprox building block presented below which allows us to efficiently and securely evaluate a public non-linear function on a secret argument, i.e., to compute $\langle \varphi'(\ell_i) \rangle$ given only $\langle \ell_i \rangle$ in this case. Note that we can compute the output of all neurons on the same layer in parallel and batch communication to increase performance. As for Hyperplane, the final step is to determine the target class $c^*$ which corresponds to the index of the neuron with the maximum output. Shares of $c^*$ are computed securely using $\text{Argmax}$ and can then be recombined by $U$, $S$, or both, as desired.

4.4.3.1 Secure Approximation of Non-linear Functions

With the state of the art STC techniques we can efficiently compute linear and polynomial functions (applying secret sharing or IIE to arithmetic circuits) or Boolean functions (applying Yao’s GCs or the GMW protocol to Boolean circuits). With these techniques we can already cover many functions used in the machine learning context, e.g., the identity, binary step, rectified linear, or maxout activation functions for neurons in an ANNs. However, a wide range of other important functions cannot be efficiently computed with the basic Boolean or Arithmetic operations such as the popular Sigmoid, Gaussian, SoftPlus, or SoftStep activation functions. In related works, such functions are often simply omitted, e.g., Dowlin et al. [DGBL+16] construct an ANN with the Sigmoid function on the output layer then note it is only important for training the network and can be left out during classification. We argue that it is generally desirable to be able to compute these functions securely as they are important building blocks for classification and other machine learning algorithms [DGBL+16, RN95].

One approach to compute these functions is to design custom tailored protocols for each of them. Custom protocols often achieve good performance and indeed we follow this approach for heavily used functions ourselves, e.g., we later design a custom protocol to evaluate a Gaussian probability density function on an additively shared point for use in our Naive Bayes classifier. However, we clearly cannot design custom protocols for each and every used function and thus argue that a general approach is desirable. To this end, we propose to evaluate functions in the above characterized category through a piecewise approximation using polynomial functions. In the most general case, given a target function $f(\cdot) : \mathbb{R} \rightarrow \mathbb{R}$ that we want to evaluate at point $x \in \mathbb{R}$, we thus aim to compute in a secure protocol

$$f'(x) = \begin{cases} g_1(x), & \text{if } x \in (-\infty, r_1) \\ g_2(x), & \text{if } x \in [l_2, r_2) \\ \vdots \\ g_k(x), & \text{if } x \in [l_k, \infty) \end{cases} \quad (4.13)$$

with $(-\infty, r_1) \cup \ldots \cup [l_k, \infty) = \mathbb{R}$ and $g_i \in \mathbb{R}[X]$ where the choice of intervals and polynomials minimizes an adequate error measure $E$, e.g., the mean-squared error or maximum distance between the true value $f(x)$ and its approximation $f'(x)$.

Protocol 4.6 presents the secure FunctionApprox protocol for this task. On the highest level, this protocol receives shares $\langle x \rangle$ of the evaluation point and the precomputed parameters of the approximation polynomials as inputs, and outputs additive
4.4. Secure Classification Framework and Designs

Input: Shared evaluation point \( \langle x \rangle \), function \( f \) held by \( U \) or \( S \)
Output: Approximated result \( \langle g_i(x) \rangle \) with \( l_i \leq x < r_i \\

Precomputation:
\( U \) or \( S \):
\( (g_1,...,g_k), (l_1,r_1,...,l_k,r_k) \leftarrow \text{Approximate}(f,k,E) \)

Parameter selection:
\( U \leftrightarrow S: \quad \tilde{x} \leftarrow \text{GC}\langle x \rangle(\langle x \rangle) \)
\( U \leftrightarrow S: \quad \tilde{a}_{l_1}^0,...,\tilde{a}_{r_1}^0 \leftarrow \text{GC}_{\tilde{x},\langle x \rangle}(\tilde{x},g_1,...,g_k,l_1,r_1,...,l_k,r_k) \)
\( U \leftrightarrow S: \quad (\tilde{a}_{l_1}^1,...,\tilde{a}_{r_1}^1) \leftarrow \text{GC}_{\tilde{x},\langle x \rangle}(\tilde{a}_{l_1}^0,...,\tilde{a}_{r_1}^0) \)

Evaluate selected polynomial:
\( U \leftrightarrow S: \quad \langle g_i(x) \rangle \leftarrow \text{EvalPoly}(\langle g_i \rangle, \langle x \rangle) \)

Protocol 4.6 The secure \textsc{FunctionApprox} protocol for the evaluation of arbitrary functions through piecewise polynomial approximation.

shares of the approximated result \( \langle f'(x) \rangle \) by first obliviously selecting the appropriate approximation polynomial \( \langle g_i \rangle = (\langle a_{l_i}^0 \rangle,...,\langle a_{r_i}^0 \rangle) \) such that \( x \in [l_i,r_i) \) and then evaluating \( g_i \) obliviously using a secure subprotocol, denoted by \( \text{EvalPoly}(\langle g_i \rangle, \langle x \rangle) \).

We explain the two most important steps, i) selecting the approximation parameters, and ii) oblivious evaluation of polynomials, in detail. Precomputing polynomials \( g_i \) that minimize the approximation error can be done on plaintexts using standard gradient descent techniques and is omitted here.

We aim to select polynomial \( g_i \) from the choice \( g_1,...,g_k \) such that \( l_i \leq x < r_i \). Since \( x \) is shared between \( U \) and \( S \), we must select \( g_i \), i.e., its coefficients \( a_{l_i}^0,...,a_{r_i}^0 \), obliviously. In order to do so, we use a selection protocol implemented as a GC which is more efficient for logical operations such as comparisons than the Arithmetic STC techniques. The corresponding Boolean circuit arranges \( k-1 \) comparators (between \( x \) and the left interval boundaries \( l_i \)) and multiplexers (to propagate selected parameters to the output gates of the circuit) in a tree of depth \( \lceil \log_2 k \rceil \). Due to the tree-wise structure of the selection circuit, it is advisable but not necessary to select the number of approximation intervals \( k \) as a power of two, since this choice maximizes the efficiency of the selection circuit. The selection circuit finally outputs \( g_i \)'s coefficients as additive shares \( \langle a_{l_i}^0 \rangle,...,\langle a_{r_i}^0 \rangle \). These shares are completely random to \( U \) or \( S \) such that neither party can determine which polynomial \( g_i \) was selected or in which interval \( [l_i,r_i) \) the secret evaluation point \( x \) lies.

Having obliviously selected the polynomial \( g_i \), we need to design a secure and efficient \( \text{EvalPoly} \) protocol that computes \( g_i(x) \) given only the shares of the polynomial \( \langle g_i \rangle \) and of the evaluation point \( \langle x \rangle \). As multiplication is interactive on additive shares, the multiplicative depth of the protocol has a higher impact on the performance of the computation than the exact amount of multiplications. We thus dismiss Horner’s method which leads to arithmetic circuits of depth \( d \). Instead, we propose a tree-based scheme that computes the desired result \( \langle g_i(x) \rangle = (a_{l_i}^0 x^d + ... + a_{k_i}^0) \) in at most \( \lceil \log_2(d) \rceil + 1 \) rounds by evaluating \( g_i(x) \) up to the term \( a_{l_i}^0 x^2 \) in round \( i \). Since we operate in our fixed-point representation (i.e., all coefficients are transformed using \( \mathbb{F}2I \)), we need one round of rescaling after each round of multiplication, resulting in a total of \( 2\lceil \log_2(d) \rceil + 1 \) communication rounds. With these techniques, we are now able to efficiently evaluate the selected polynomial \( g_i \) of degree \( d \) at a secret point \( x \), denoted by \( \langle g_i(x) \rangle \leftarrow \text{EvalPoly}(\langle g_i \rangle, \langle x \rangle) \).
Input: $U$ has feature vector $\vec{x}$

$S$ has Naive Bayes classification model $M = (P(X|C), P(X), P(C))$

Output: Class $c^* = \text{NaiveBayes}(M, \vec{x})$

Initialize:

$S: \langle \hat{p}(c_j) \rangle_S = \langle f2li(p(c_j)) \rangle \forall j = 1...k$

$U: \langle \hat{p}(c_j) \rangle_U = \text{Logzero} \forall j = 1...k$

Compute posteriors:

$U \leftrightarrow S: \langle \hat{p}(x_i|c_j) \rangle \leftarrow \text{EvalProb}(x_i, P(X_i|c_j)) \forall i = 1...n, \forall j = 1...k$

$U, S: \langle \hat{p}(c_j|\vec{x}) \rangle = \sum_{i=1}^{n} \langle \hat{p}(x_i|c_j) \rangle + \langle \hat{p}(c_j) \rangle \forall j = 1...k$

Determine most probable class:

$U \leftrightarrow S: \langle c^* \rangle \leftarrow \text{Argmax}(\langle \hat{p}(c_j|\vec{x}) \rangle)$

$U \leftrightarrow S: c^* \leftarrow \text{Recombine}(\langle c^* \rangle)$

Protocol 4.7 The secure NaiveBayes classifier protocol.

To conclude, we note that the function $f$ needs to be known only by the party who precomputes the polynomials $g_i$. The other party, in contrast, remains oblivious of the function that is approximated since it does not learn the approximation parameters (except for the degree and number of the used polynomials). Applying FUNCTIONAPPROX to ANN thus allows hiding the activation functions $\phi^1, ..., \phi^L$ from $U$, e.g., when those functions represent private knowledge to $S$ (cf. Section 2.2.2).

4.4.4 Naive Bayes

We now turn to a different breed of classifiers, those based on the Bayes theorem with assumptions of strong conditional independence as introduced in Section 4.2.1. Different than the previous two classifiers, Naive Bayes classifiers are constructed over probability distributions which will bring additional challenges in the context of secure computations, especially in terms of accuracy, which will prepare us for the even more challenging Forward and Viterbi algorithms on HMMs that involve computations over very small probabilities. Since Naive Bayes are among the most widely used classifiers in practice, efficient secure protocols for them also represent a valuable contribution on its own.

Naive Bayes classifiers select the target class according to the posterior probabilities $p(c_j|\vec{x})$ (Equation 4.5), which are computed as the product of the feature likelihoods $p(x_i|c_j)$ (Equation 4.7). Since secure multiplications are expensive and introduce numerical errors, we transform computations of the posteriors into logspace:

$$\log(p(c_j|\vec{x})) = \sum_{i=1}^{n} \log(p(x_i|c_j)) + \log(p(c_j)) - \log(p(x_i))$$ (4.14)

The representation given by Equation 4.14 is advantageous since it contains only additions which can be computed much more efficiently and accurately over additive shares. Note that even on plaintexts, Naive Bayes is often computed in logspace to increase the numerical stability.

Protocol 4.7 provides a protocol for a secure Naive Bayes classifier in this logspace representation. At the start, $U$ holds the feature vector $\vec{x}$ and $S$ has the Naive Bayes
classification model consisting of the probability mass functions $P(X|C)$, $P(X|C)$. $S$ first transforms the priors $p(c_j)$ and likelihoods $p(x_i|c_j)$ to scores with fixed-point precisions using 2li and both parties initialize shares of the priors $(\hat{p}(c_j))$ using dummy sharing. In the next step, $U$ and $S$ evaluate the probability measure $P(X_i|c_j)$ known only to $S$ for the event $X_i = x_i$ known only to $U$ using the secure EvalProbMass protocol presented below. Given the shared likelihoods $(\hat{p}(x_i|c_j))$, computing shares of the posteriors $(\hat{p}(c_j|x))$ is a simple matter of summing the shares of the likelihoods and the priors. Finally, we determine shares of the class that maximizes the posterior scores and let $U$, $S$, or both recombine the result as desired. Note that we drop the evidence $p(x_i)$ since it is constant for a fixed feature vector $\bar{x}$ and thus only linearly scales the scores $\hat{p}(c_j|x)$ which does not change the argmax $c^*$.  

### 4.4.4.1 Secure Evaluation of Probability Mass and Density Functions

Our secure Naive Bayes classifier above requires $U$ and $S$ to evaluate a probability measure $P(X_i|c_j)$ known only to $S$ for an event $x_i$ known only to $U$ with the result returned as an additive sharing in logspace. $(\hat{p}(x_i|c_j))$. The same primitive will be later required to determine emission probabilities during Forward and Viterbi computation on HMMs. In the following, we present two secure protocols for this task, beginning with the case where $X_i$ corresponds to a discrete feature such that $P(X_i|c_j)$ is a probability mass function and, afterwards, handling the second case where features are continuous and $P(X_i|c_j)$ is a probability density measure. We simplify and generalize notation by assuming a probability distribution in one random variable $P(X)$ which is conceptually the same as $P(X_i|c_j)$ for a fixed $c_j$.

#### Evaluating probability mass functions

Protocol 4.8 provides the details of the EvalProbMass protocol for the discrete case. Note that we assume for simplicity that the feature $X$ has only a finite amount of possible outcomes $\Omega = \{x_1, \ldots, x_n\}$. In the first step, $S$ transforms the probability mass function $P(X)$ into our logspace fixed-point number representation using 2li. To hide the plaintext values, $S$ blinks each component with the same random value $r_S \in \mathbb{Z}_2$. Both parties then engage in $\text{1-m-OT}_U$ on the $m$ possible l bit scores in $P(X)$ after which $U$ obtains the blinded score $\hat{p}(x_i) + r_S$. The use of OT guarantees that $U$ learns only $\hat{p}(x_i) + r_S$ and that $S$ learns nothing about the feature value $x_i$. Note that $(\hat{p}(x_i|c_j)) = (\hat{p}(x_i) + r_S, -r_S)$ already is the desired additive sharing of the score $\hat{p}(x_i|c_j)$.

In NaiveBayes above, EvalProb is invoked for each of the $k$ classes $c_j \in C$. Since the choice $x_i$ is always the same, we can efficiently realize these $k$ calls in a single $\text{1-m-OT}_U$, where the $r$th secret is the concatenated bitstring $\hat{p}(x_i), \ldots, \hat{p}(x_i|c_j)$ of length $kl$ bit. Additionally, this is repeated for each of the $n$ features $X_1, \ldots, X_n$, and we can execute all OTs in parallel to further increase performance resulting in
one call to 1-m-OT\(_n\). The structure of Forward and Viterbi allows a very similar optimization as we explain later in Section 4.4.5 and 4.4.6.

In related work [PDRO10, FDH+11], the same problem is often solved using HE, i.e., \(U\) encodes and encrypts the choice \(x_i\) in \(m\) selectors of which only the \(i\)th ciphertext encrypts a one and all others are encryptions of zero, i.e., \([0]...[1]...[0]\). \(S\) can then obtain an encryption of \(p(x_i | c_j)\) by multiplying these selectors pairwise against the values in \(P(X_i | c_j)\) and summing the results using the homomorphic properties of the HE scheme, i.e., \([0] \odot p(x_1 | c_j) \oplus ... \odot [1] \odot p(x_m | c_j) \oplus ... \odot [0] \odot p(x_m | c_j)\). In comparison, our scheme is much more efficient as it is based on OT which can be realized using a combination of highly efficient symmetric key cryptography primitives and one-time-pad operations (cf. Section 2.3.1).

Finally, we note that in our and related approaches, the event \(x_i\) and probability mass function \(P(X_i | c_j)\) must be known in clear by \(U\) and \(S\), respectively. Extending our approach to a setting where \(P(X_i | c_j)\) is shared over the two parties is straightforward, while the case where also \(x_i\) is shared by both parties is more involved. Since we require neither variant for our secure classifiers, we omit them here.

**Evaluating probability density functions.** We now consider the case where \(X_i\) is a continuous feature with the set of possible outcomes \(\Omega = \mathbb{R}\). A simple first approach would be to take a representative finite set of samples from \(P(X_i | c_j)\) and then invoke \textsc{EvalProbMass} on it. Of course the set of samples has to be large enough to provide reasonable accuracy. Indeed, we find in our evaluation of the building blocks (Appendix A) that \textsc{EvalProbMass} can efficiently handle thousands of probability masses such that this approach is viable.

Another option is to use the previously proposed \textsc{FunctionApprox} protocol to approximate the probability density function. This approach is practical and handles arbitrary probability densities. However, in our design of \textsc{FunctionApprox} the evaluation point \(x\) is shared between both parties which is not required here since \(U\) knows it in clear. Thus, the use of \textsc{FunctionApprox} may introduce unnecessary overheads and we consider the design of more efficient custom protocols. In the following, we demonstrate how this can be achieved by the example of the Gaussian distribution, whose frequent use justifies the additional handwork.

For a feature with Gaussian (Normal) distribution \(X_i \sim \mathcal{N}(\mu_i, \sigma_i)\), we define

\[
p(x_i) = \frac{1}{\sqrt{2\pi \sigma_i^2}} e^{-\frac{1}{2} \left( \frac{x_i - \mu_i}{\sigma_i} \right)^2} (4.15)
\]

We note that this is incorrect in the strict sense of probability theory, since \(p\) is a density function that has to be integrated over a certain interval to obtain a correct probability. However, defining \(P(X_i)\) as above has proved practical in the context of classification [VLW+12] as it can be evaluated very efficiently in logspace:

\[
\log(p(x_i)) = \log\left( \frac{1}{\sqrt{2\pi \sigma_i^2}} \right) + \frac{(x_i - \mu_i)^2}{-2\sigma_i^2} (4.16)
\]

As we will see next, Equation 4.15 can also be efficiently evaluated in a secure protocols and we thus tolerate the slight lapse in probability theory.
protocols carried out by we substitute those steps that require both parties’ inputs with secure interactive and Viterbi algorithms on functions. We are now ready to tackle the most complex classifier, the Forward for numerical accuracy and developed efficient primitives for evaluating probability Naive Bayes classifiers, we repeatedly took advantage of computations in logspace revealed the need for secure function approximation. For our designs of secure for the computation of scalar products and argmax, while the extension to ANNs for HMMs which will require a combination of all developed techniques and gained insights to create accurate and efficient secure protocols.

Our secure FORWARD (Protocol 4.10) computes exactly the same initialization, recursion, and termination steps as the classical Forward algorithm (cf. Section 4.3). However, to achieve sufficient accuracy, we transform all computations into logspace, i.e., a product of probabilities becomes a sum of scores and a sum of probabilities becomes a logsum of scores. Furthermore, to keep the inputs of both parties private, we substitute those steps that require both parties’ inputs with secure interactive protocols carried out by U and S.
Input: $U$ has $O \in V^{1 \times T}$, $S$ has $\lambda = \{O, V, A, B, \pi\}$

Output: Viterbi score $\hat{P}(O, S^\ast | \lambda)$ and Viterbi path $S^\ast \in S^{1 \times T}$

Initialization: For $1 \leq i \leq N$

$U \Rightarrow S$ : $\langle \hat{b}_i(0) \rangle \leftarrow \text{EvalProbMass}(a_i, B_i)$
$U, S$ : $\langle a_i(0) \rangle = \langle \hat{b}_i(0) \rangle \oplus \langle \hat{a}_i \rangle$
with $\langle \hat{a}_i \rangle_U = \text{Logzero}$ and $\langle \hat{a}_i \rangle_S = \#2li(\pi_i)$

Recursion: For $2 \leq t \leq T, 1 \leq i \leq N$

$U \Rightarrow S$ : $\langle \hat{b}_i(0) \rangle \leftarrow \text{EvalProbMass}(a_i, B_i)$
$U, S$ : $\langle g^*_t(i) \rangle = \langle \hat{a}_i(1) \rangle \oplus \langle \hat{a}_i(2) \rangle \oplus \cdots \oplus \langle \hat{a}_i(N) \rangle \oplus \langle \hat{a}_i \rangle$
with $\langle \hat{a}_i \rangle_U = \text{Logzero}$ and $\langle \hat{a}_i \rangle_S = \#2li(\pi_i)$
$U \Rightarrow S$ : $\langle \hat{a}_i(1) \rangle \leftarrow \text{LogSum}(\langle g^*_t(i) \rangle)$
$U, S$ : $\langle a_i(1) \rangle = \langle \hat{a}_i(1) \rangle \oplus \langle b_i(0) \rangle$

Termination:

$U, S$ : $\langle \hat{a}_T \rangle = \langle \hat{a}_T(1) \rangle \oplus \cdots \oplus \langle \hat{a}_T(N) \rangle$
$U \Rightarrow S$ : $\langle \hat{P}(O, S^\ast | \lambda) \rangle \leftarrow \text{LogSum}(\langle \hat{a}_T \rangle)$
$U \Rightarrow S$ : $\hat{P}(O | \lambda) \leftarrow \text{Recombine}(\langle \hat{P}(O | \lambda) \rangle)$

Protocol 4.10 The secure Forward protocol: $U$ holds an observation sequence $O$ that she wants to match against the HMM $\lambda$ held by $S$. Using our secure EvalProbMass and LogSum primitives, the parties compute shares $\langle \hat{P}(O | \lambda) \rangle$ of the Forward score securely, i.e., without either party revealing their sensitive input to the other. The result final result $\hat{P}(O | \lambda)$ is opened to $U$, $S$, or both parties as desired.

On the highest level, we only need two secure protocols, EvalProbMass and LogSum, to build Forward. EvalProbMass is used to share emission scores additively between $U$ and $S$ without either party learning the other’s input, i.e., the observations $o_1, ..., o_T$ held by $U$ or the emission probability matrix $B$ held by $S$. LogSum extends our portfolio of secure primitives by a secure protocol for computing shares of the logarithm of the sum of two probabilities that are given only as additive shares in logspace. In the following, we explain how $U$ and $S$ compute each of the three phases (initialization, recursion, and termination) in more detail.

Initialization: At the start, $U$ holds an observation sequence $O = o_1, ..., o_T$ and $S$ holds the HMM $\lambda$. The goal of the initialization step is to compute additive shares of the forward variables $a_i(t) = \pi_i \cdot b_i(o_t)$ in logspace:

$$\hat{a}_i(t) = \hat{a}_i + \hat{b}_i(o_t) \quad \forall i = 1 ... N \quad (4.17)$$

To compute the desired shares $\langle \hat{a}_i(t) \rangle$, $U$ and $S$ first invoke EvalProbMass $N$ times after which both parties obtain additive shares of the emission scores $\langle \hat{b}_i(o_t) \rangle \forall i = 1 ... N$. $S$ then adds the prior state scores $\hat{a}_i$ and $U$ adds Logzero.

Recursion: The goal of the recursion step is to compute additive shares of the forward variables $a_i(t) = \prod_{j=1}^{t-1} (a_i(j) \cdot a_{j+1}) \cdot b_i(o_t)$ in logspace

$$\hat{a}_i(t) = \log \left( \sum_{j=1}^{t-1} (a_i(j) \cdot a_{j+1}) \right) + \hat{b}_i(o_t) \quad \forall i = 1 ... N \quad (4.18)$$

given only the additively shared forward scores $\langle a_{i-1}(j) \rangle$ from the previous iteration.
4.4. Secure Classification Framework and Designs

To compute Equation 4.18 securely on additive shares, \( U \) and \( S \) first invoke EvalProbMass to additively share the emission scores \( \hat{b}_i(\alpha_t) \) as before. Implementing Forward, we actually batch all calls to EvalProbMass into one invocation of 1-M-OT\( N_l \) to increase performance as done before in our design of the Naive Bayes classifier on discrete features (cf. Section 4.4.4.1).

Having shared the emission scores, both parties now use our secure LogSum primitive (presented below) to compute additive shares of the partial forward scores \( \hat{\alpha}_t'(i) = \log \left( \sum_{j=1}^{N} \alpha_{t-1}(j) a_{ji} \right) \). Prior works [SS07, PRSR11, PRRS13] transform back to the probabilities \( \alpha_{t-1}(j) \) and \( a_{ji} \) and multiply them, which is liable to cause critical underflows for medium to large problem instances. To avoid these inaccuracies (as well as the conversion overheads), we instead compute a piece-wise linear approximation (PLA) of the logsum function based on our previous FunctionApprox protocol which operates only in logspace. The result of LogSum is again distributed as additive shares between \( U \) and \( S \) and they only need to locally add their shares \( \langle \hat{b}_i(\alpha_t) \rangle \) of the emission scores to obtain the desired additive sharing \( \langle \hat{\alpha}_t(i) \rangle \) (cf. Equation 4.18).

**Termination:** In the termination step, we must again compute a logsum of the forward variables to obtain the final result

\[
\hat{p}(O|\lambda) = \log \left( \sum_{j=1}^{N} \alpha_T(i) \right)
\]  

(4.19)

Since the forward variables are given only as logspaced additive shares \( \langle \alpha_T(i) \rangle \), we need to employ the secure LogSum primitive again to compute Equation 4.19. As the final result, the two parties \( U \) and \( S \) each hold additive shares of the Forward score, \( \langle \hat{p}(O|\lambda) \rangle_U \) and \( \langle \hat{p}(O|\lambda) \rangle_S \), respectively. Depending on who should learn the result in the concrete use case, the parties exchange their shares to enable reconstruction of the desired Forward score \( \hat{p}(O|\lambda) \).

**4.4.5.1 Secure Logsum Primitive**

When we used logspace transformations during our design of secure Naive Bayes classifiers, we only had to compute additions in logspace since Naive Bayes only uses multiplications in probability space. However, when an algorithm performs additions in probability space, these translate to the logarithm of a sum in logspace that cannot be further simplified. This operation is referred to as a logsum and is ubiquitous not only in HMM computations but also in signal processing and pattern classification in general [PRT15].

Concretely, we need to compute logsums during the recursion step of the logspace Forward, i.e., \( \hat{\alpha}_t'(i) = \log \left( \sum_{j=1}^{N} \alpha_{t-1}(j) a_{ji} \right) \). Since, the probabilities \( \alpha_{t-1}(j) \) and \( a_{ji} \) are shared in their corresponding logspace representation between \( U \) and \( S \), we require a secure LogSum protocol that operates over additive shares and returns the result in shared form. Computing logsums is the main performance bottleneck of Forward as it has also been for previous approaches [FDH+11]. Due to its high performance impact, we thus explain our approach in detail here which is based on FunctionApprox but optimized further for performance.
Input: Two shared summands \((\hat{x}, \hat{y})\), PLA parameter \(k \in \mathbb{N}\)

Output: Additive sharing

\[ \log(x + y) = \max(x, y) + \log(1 + \exp(\min(x, y) - \max(x, y))) \]  

Equation 4.20 could be computed using the secure floating point primitives from [ABZ13, DDK+15] or using HE and fixed-point precision with rescaling as proposed in [PRRS13, SS07]. Franz et al. [FDH+11] compute Equation 4.21 through oblivious lookup tables which grow exponentially in the bitlength of the inputs and fresh tables have to be transferred for each logsum operation. We deem these approaches too expensive for our use case and follow the alternative idea of Portelo et al. [PRT15] to compute a PLA of Equation 4.20. Our LogSum protocol reuses their Boolean circuit design to select the applicable approximation parameters, but embeds it in a hybrid protocol that efficiently combines GCs with additive sharings, in contrast to the completely GC-based solution of proposed by Portelo et al. [PRT15]. This not only improves performance but, more importantly, allows us to compose LogSum with our other secure (sub-)protocols that operate over arithmetic shares.

The details of our own approach, LogSum, are given in Protocol 4.11. In a precomputation step (that can happen at any time and needs to be computed only once), \(S\) computes the parameters for the PLA. \(S\) selects \(k\) intervals \([l_i, r_i]\) \(0 \leq i \leq k-1\) and computes a linear regression \(g_i(x) = m_i x + n_i\) of \(\log(1 + \exp(-d))\) for each interval \([l_i, r_i]\). To optimize performance and accuracy, we note that for double precision it is enough to restrict the considered range for \(d\) to \([0, 0.16]\) since \(d = \max(x, y) - \min(x, y)\) and for \(d \geq 16\) we have \(\max(x, y) > 10^{16} \cdot \min(x, y)\) while a double precision mantissa of 53 bit can only encode approximately 16 decimal digits.

In the first protocol step, \(U\) and \(S\) convert their additively shared inputs into garbled inputs by evaluating a garbled addition circuit [DSZ15] to which each party inputs their respective shares \(\langle \hat{x} \rangle\) and \(\langle \hat{y} \rangle\). As the result, the values \(\hat{x}\) and \(\hat{y}\) are available.
Input: $U$ has $O \in V^{1 \times T}$, $S$ has $\lambda = \{S, V, A, B, \pi\}$
Output: Viterbi score $\hat{P}(O, S'|\lambda)$ and Viterbi path $S' \in S^{1 \times T}$

Initialization: For $1 \leq i \leq N$
\[ U \Rightarrow S': \hat{b}_i(n_i) \leftarrow \text{EvalProbMass}(a_i, R_i) \]
\[ U, S: \hat{o}_i(j_i) = \hat{b}_i(o_i) + (\hat{v}_i) \]
with $(\hat{u}_i,v_i) = \text{Logzero}$ and $(\hat{u}_i,v_i) = \text{F2LI}(\pi_i)$

Recursion: For $2 \leq t \leq T, 1 \leq i \leq N$
\[ U \Rightarrow S': \hat{o}_i(n_i) \leftarrow \text{EvalProbMass}(a_i, R_i) \]
\[ U, S: \hat{g}_i(o_i) = \{\hat{o}_i(1) + \hat{o}_i, \ldots, \hat{o}_i(N) + \hat{o}_i\} \]
with $(\hat{u}_i,v_i) = \text{Logzero}$ and $(\hat{u}_i,v_i) = \text{F2LI}(\pi_i)$
\[ U \Rightarrow S': \hat{g}_i(o_i) \leftarrow \text{MaxArgmax}(\hat{g}_i(o_i)) \]
\[ U, S: \hat{o}_i(j_i) = \hat{o}_i(j_i) + \hat{b}(a_i) \]

Termination:
\[ U, S: \hat{g}_T = \{(\hat{o}_i(1)) \ldots, (\hat{o}_i(N))\} \]
\[ U \Rightarrow S': \hat{P}(O, S'|\lambda), \hat{r} \leftarrow \text{MaxArgmax}(\hat{g}_T) \]
\[ U \Rightarrow S: \hat{P}(O, s_t^*), \hat{s}_{t+1} \leftarrow \text{RECOMBINE}(\hat{P}(O,\lambda), (\hat{s}_{t+1})) \]

Backtracking: For $T \geq t \geq 2$
\[ U \Rightarrow S: (\hat{s}_{t-1})_x \leftarrow 1\text{-OT}(s_t^*; (\hat{o}_i(1))_x, \ldots, (\hat{o}_i(N))_x) \]
\[ U \Rightarrow S: \hat{s}_{t-1} \leftarrow \text{RECOMBINE}(\hat{s}_t) \]

Protocol 4.12 The secure VITERBI protocol: $U$ holds an observation sequence $O$ that she wants to match against the HMM $\lambda$ held by $S$. Using our secure EvalProbMass and MaxArgmax primitives, the parties compute shares $\hat{P}(O,\lambda)$ of the Viterbi score securely, i.e., without either party revealing their sensitive input to the other. The result final result $\hat{P}(O,\lambda)$ and final state $s_t^*$ is opened to $U$, $S$, or both parties as desired. Tracking back from $s_t^*$, the Viterbi path $S'$ is reconstructed by $U$ or $S$ using OT.

in garbled form for further processing in the subsequent GC. Both parties then evaluate the first part of Portelo’s selection circuit [PRT15], $C_{\text{Select}}$, which obliviously computes $\max(\hat{x}, \hat{y})$ and $d = |\hat{x} - \hat{y}|$ and then obliviously selects the parameters of the regression line $y$, where $\hat{t}_i \leq d < \tau_i$. We now convert the approximation parameters $n, m$ as well as the $\max(\hat{x}, \hat{y})$ and distance $d = |\hat{x} - \hat{y}|$ back to additive shares using the OT-based subtraction protocol proposed in [DSZ15] (to convert a garbled $\hat{x}$ to a sharing $(\hat{x})$, we compute $\hat{x} - r$ in the GC and output shares $(\hat{x}) = (\hat{x} - r, r)$). Finally, we derive shares of the final result $\max + m \cdot d + n$ using the corresponding addition and multiplication operations over additive shares.

### 4.4.6 HMM Viterbi

The Viterbi and Forward algorithm are principally very similar: Both algorithms are initialized in exactly the same way and compute the desired result using dynamic programming over the forward variables $\alpha_i(t)$. However, where the Forward

---

2The variable names can be confusing: their name is not derived from the Forward algorithm but relates to the procedure of going forward through the observation sequence. The Forward-Backward algorithm, e.g., goes forward then backward through the observations and has corresponding forward and backward variables. The Viterbi algorithm goes forward through the observations and thus also has forward variables which are, however, computed differently than those in the Forward algorithm.
algorithm computes a sum over the probabilities of all state sequences up to time $t$ (cf. Equation 4.18) the Viterbi algorithm tries to determine the single best path and hence computes the maximum over the probabilities of all previous state sequences up to that time. A second difference is that the Viterbi algorithm needs to maintain a backtracking matrix of the maximum arguments to be able to reconstruct the most likely path in an additional fourth phase after the main algorithm has finished.

The details of our secure Viterbi protocol, which computes exactly the same steps as the Viterbi algorithm on unencrypted data, are given in Protocol 4.10. On the highest level, we only need two secure protocols, EvalProb and MaxArgmax. EvalProb is used in exactly the same way as for Forward, while MaxArgmax substitutes the LogSum primitive. Additionally, we need to keep the backtracking matrix composed of the variables $\upsilon_t(i)$, $\forall t = 1...T$, $\forall i = 1...N$. In the following, we explain in more detail how $U$ and $S$ compute each of the four phases (initialization, recursion, termination, and backtracking).

**Initialization:** Viterbi is initialized exactly as Forward, i.e., by sharing the emission scores $\hat{b}_i(o_1)$ via EvalProb and adding the initial state scores locally.

**Recursion:** The goal of the recursion step is to compute the forward variables $\hat{\alpha}_t(i)$ in logspace, i.e., the probability of the optimal partial state sequence given only the partial observation sequence $o_1o_2...o_t$ up to time step $t$. Additionally, we need to keep track of this optimal state sequence in the variables $\upsilon_t(i)$. These steps are given in logspace by the following two equations:

$$\hat{\alpha}_t(i) = \max_{s_j \in S} (\hat{\alpha}_t(j) + \hat{b}_i(o_t)) \quad \forall i = 1...N$$  \hspace{1cm} (4.22)

$$\upsilon_t(i) = \arg \max_{s_j \in S} (\hat{\alpha}_t(j) + \hat{b}_i(o_t)) \quad \forall i = 1...N$$  \hspace{1cm} (4.23)

We note that max and argmax have exactly the same range $s_j \in S$ and argument $\hat{\alpha}_t(j) + \hat{b}_i(o_t)$. In our secure protocol, we thus combine both into one primitive, MaxArgmax, to gain performance. MaxArgmax operates over additive shares and returns the result again in shared form to $U$ and $S$ who only need to locally add their shares $\langle \hat{b}_i(o_t) \rangle$ of the emission scores to obtain the desired additive sharing $\langle \hat{\alpha}_t(i) \rangle$ of the forward scores $\hat{\alpha}_t(i)$ (Equation 4.22). Since we also obtain additive shares of the maximum argument from MaxArgmax, we can directly set the backtracking variables $\upsilon_t(i)$ (Equation 4.23).

**Termination:** We compute the Viterbi score $\hat{P}(O,S^*|\lambda)$ and the final state $s_T^*$.

$$\hat{P}(O,S^*|\lambda) = \max_{s_{T}} (\hat{\alpha}_{T}(j))$$  \hspace{1cm} (4.24)

$$s_{T}^* = \arg \max_{s_{T}} (\hat{\alpha}_{T}(j))$$  \hspace{1cm} (4.25)

As during recursion, we combine the max and arg max functions to gain performance. Using the secure MaxArgmax primitive, the two parties $U$ and $S$ each obtain additive shares $\langle \hat{P}(O,S^*|\lambda) \rangle$ and $\langle s_T^* \rangle$ of the Viterbi score (Equation 4.24) and the final state $s_T^*$ (Equation 4.25), respectively. Depending on who should learn the result, the parties exchange their shares to enable reconstruction of the score and final state. In order to reconstruct the most probable state sequence (the Viterbi path) that led to $s_T^*$, we need to additionally execute the following fourth phase.
4.4 Secure Classification Framework and Designs

Backtracking: Remember that \(\upsilon_t(i)\) stores the most likely predecessor of state \(s_i\) given \(o_1o_2...o_t\). To construct the most probable path through the internal states of the HMM given the observation sequence, we thus start with the most probable final state \(s^*_T\) (Equation 4.25) and go backwards in time:

\[
s^*_{t-1} = \upsilon_t(s^*_t)
\]

Of course, the challenge is to compute Equation 4.26 obliviously with the backtracking matrix \(\upsilon\) given only as additive shares distributed among \(U\) and \(S\). The steps in Protocol 4.12 show how this is achieved in the case where \(U\) should learn the path but not \(S\) (the opposite case is symmetric). Given the starting point \(s^*_T\), \(U\) obtains \(S\)'s share \(\langle s^*_T \rangle\) through 1-out-n-OT which guarantees that \(S\) does not learn the choice \(s^*_T\) and that \(U\) learns nothing except the desired share. \(U\) can then recombine \(s^*_{T-1}\) and iteratively repeat those steps until \(U\) obtains the complete Viterbi path \(S^* = s^*_1,...,s^*_T\). Note that we can realize backtracking without OT when both parties should learn the Viterbi path: \(U\) and \(S\) just recombine \(s^*_T\), exchange shares \(\langle \upsilon_T - 1(s^*_T) \rangle\), and repeat until reaching \(s^*_1\).

We have now presented the secure protocol designs that make up our SHIELD framework for privacy-preserving machine learning and pattern recognition, the first main contribution in this chapter. In the following sections, we show that our designs fulfill the security, accuracy, and performance requirements of our problem statement (Section 4.3.1). The fourth requirement, customizing to mobile users and constrained environments, will be addressed afterwards in the second main part of this chapter as this requires different optimization strategies and significant extensions to our protocol designs so far.

4.4.7 Security Discussion

We now show that our classifiers are secure in the semi-honest adversary model. For the security proofs of the basic STC techniques that underlie our approaches, i.e., OT, additive sharings, and GCs, we refer to Section 2.3. We begin by showing that neither party learns anything in our proposed building blocks protocols, i.e., neither from the inputs, nor the outputs, nor any intermediate values. We then invoke Canetti’s modular sequential composition theorem [Can00] to argue that our classifier designs built on top of these primitives are secure.

4.4.7.1 Security of the Building Blocks

Security of Rescale. To argue security of RESCALE (Protocol 4.1, Section 4.4.1.1), we show that neither party learns anything about the value \(x\) that is rescaled. First, note that the shares \(\langle x \rangle_U\) and \(\langle x \rangle_S\) are random and a single share conveys no information about \(x\) to either party. During the protocol, \(U\) blinds her share \(\langle x \rangle_U\) by a \(l + \kappa\)-bit random number \(r\) and sends it to \(S\). \(S\) learns \(\langle x \rangle_U + \langle x \rangle_S = x + r\). Blinding with \(r\) over \(Z\) achieves statistical security towards \(S\) with security parameter \(\kappa\). Since \(U\) receives no messages, RESCALE is unconditionally secure against \(U\).

Security of ScalarProduct. SCALARPRODUCT (Protocol 4.3, Section 4.4.2.1) follows from the security of addition and multiplication over additive sharings as discussed in Section 2.3.3.1.
Security of MaxArgmax. All steps of MAXARGMAX are realized in one monolithic GC – we emphasize that we differentiate the three steps in our protocol description only for reasons of clarity but implement them in one single GC which yields better performance. Consisting of only one GC, security for these steps follows directly from the security of Yao’s GCs protocol (cf. Section 2.3.2.1). The inputs and outputs are all additively shared over both parties which are perfectly random and thus reveal no information to either party holding only a single share.

Security of EvalProbMass. To show that EvalProbMass (Protocol 4.8, Section 4.4.4.1) is secure, we argue that $U$ does not learn anything about the probability mass function $P(X)$ held by $S$ and $S$ does not learn anything about the event $X = x_i$ held by $U$. $S$ first blinds all probabilities additively over $\mathbb{Z}_2^l$ with the random value $r_S \in \mathbb{Z}_2^l$. From $U$’s perspective, the blinding represents a One-Time-Pad encryption which is unconditionally secure. Since all emission scores are blinded by the same random value $r_S$, we use 1-out-of-2-OT to guarantee that $U$ learns only exactly one blinded emission score. $U$ then arithmetically shares the value with $S$, which does not leak information as the employed additive sharing uses perfect blinding over $\mathbb{Z}_2^l$. Finally, $S$ subtracts $r_S$ from its share, which is a local operation and reveals no information. Hence, EvalProb is secure in the semi-honest model.

Security of Gaussian. GAUSSIAN (Protocol 4.9, Section 4.4.4.1) only composes secure additions, multiplications, and the Rescale protocol already discussed above. We thus conclude that it is secure in the semi-honest model.

Security of FunctionApprox and LogSum. LogSum (Protocol 4.11, Section 4.4.5) is an extension of FunctionApprox (Protocol 4.6, Section 4.4.3.1). It is thus sufficient to show that LogSum is secure, i.e., that neither party learns anything about the summands $\hat{x}$ and $\hat{y}$ and $\hat{z}$.

We first note that the summands $\hat{x}$ and $\hat{y}$ are given as additive shares $\langle \hat{x} \rangle$ and $\langle \hat{y} \rangle$ and a single share is completely random and does not reveal any information to its holder. Further, the PLA parameters $k$ and $\mathcal{P}$ are completely independent of the inputs and thus reveal no information about them either.

The following three protocol steps, involving i) input conversion, ii) the selection of approximation parameters, and iii) the conversion of outputs, are realized in one monolithic GC. As for MAXARGMAX before, we only differentiate these three steps in our protocol description for reasons of clarity but implement them in one single GC which yields better performance. Consisting of only one GC, security for these steps follows directly from the security of Yao’s GCs protocol (cf. Section 2.3.2.1). The output of these steps is additively shared over both parties, i.e., $\langle \text{max} \rangle$, $\langle d \rangle$, $\langle m \rangle$, and $\langle n \rangle$, which reveals no information to either party holding only a single share of each output since additive sharing implements perfect blinding over $\mathbb{Z}_2^l$. It is also important to note that the structure of the circuit is independent of all parameters except for the public parameter $k$, therefore leaking no sensitive information.

In the penultimate step, we securely compute the product over additive shares and apply Rescale on it. All outputs are again additively shared and reveal no information to either party. The last step involves an addition operation over additive shares which is executed locally and has no security implications in the semi-honest model. Finally, the output $z$ is obtained by the two parties in shared form, where a single share is indistinguishable from a random value and reveals no information.
In summary, security of LogSum depends on the security of Yao’s GCs and the Rescale protocol. As Rescale offers only statistical security against a semi-honest $S$, LogSum as well offers only statistical security.

### 4.4.7.2 Security of the Classifier Designs

The security argument is the same for all our classifier designs, i.e., Hyperplane, ANN, NaiveBayes, Forward, Viterbi. We argue that $U$ learns nothing about the involved classification model $M$ or HMM $\lambda$ (private input of $S$), and, vice versa, $S$ learns nothing about the feature vector $\vec{x}$ or observation sequence $O$ ($U$’s private input), except of course for what is implied in the final result that is learned in clear by one or both parties. This proposition holds since any interaction between $U$ and $S$ in all our secure classifiers happens only through one or multiple of the primitives discussed above. As we have showed in the previous section, all of these primitives are secure in the semi-honest model and return their output in the form of random additive sharings such that their output does not reveal anything to either party. In other words, their use reveals no information about the inputs or any intermediate values. This allows us to compose them and the composition is then secure according to the modular sequential composition theorem for semi-honest protocols [Can00].

All other steps in the secure classification protocols are local operations that have no security implications in the semi-honest model. Finally, one or both parties learn the output by recombining the shared result which is of course as intended.

It is important to note that the utilized STC techniques protect the inputs (i.e., the models and feature vectors) but not the structure of the evaluated classification function. In particular, this implies that $S$ learns the length of the feature vector $\vec{x}$ or observation sequence $O$ while $U$ learns the dimension of the models, e.g., the total number of possible classes, the number of layers and neurons in an ANN, or the number of states and possible emissions in an HMM. We emphasize that this is fully within the security model defined in Section 2.2. If desired this can be prevented in all our designs by padding inputs with dummy features or observations and models with dummy weights, neurons, states, and so forth but this inevitably increases processing and communication overheads. Another approach are universal circuits that also hide the function that is being evaluated [KS08b, KS16] (i.e., enabling Private Function Evaluation (PFE) as introduced in Section 2.2.2). PFE causes orders of magnitude higher overheads than the secure evaluation of public functions. We argue that the costs of PFE are not justified in our application context since our classification algorithms are publicly known and do not require protection.

### 4.4.8 Evaluation

To thoroughly quantify the performance and accuracy of our approaches, we implement prototypes of all our secure classifier and evaluate them on real-world datasets widely used in the machine learning community (a brief complementary evaluation of the underlying secure building blocks is given in Appendix A). We compare the performance of each of our secure classifiers against the respective fastest approach in the related work and point out our significant improvements.
Implementation. We implement all secure primitives and classifiers prototypically in C++. EvalProb and Viterbi require the 1-\(\eta\)-OT\(^n\) primitive which we implement as one invocation of 1-2-OT\(^{\log_2(n)}\) according to [NP01,NP05], employing the efficient OT extensions described in [ALSZ13]. FunctionApprox, LogSum and MaxArgmax require creating and evaluating GCs for which we build upon the ABY framework for STC [DSZ15]. Protocols for multiplication on additive shares are also available in ABY. Besides the OT extensions and the ABY framework, which are fully multithreaded, the rest of our implementation realizes only obvious optimizations, e.g., batching of the operations in the inner for loop of the Forward algorithm as indicated in the design section. Further optimizations are conceptionally possible, e.g., pipelining GC generation and evaluation as proposed in [HEKM11], but were not realized as they would have required substantial modifications of the ABY runtime environment.

Experimental Setup. We perform all experiments between two standard desktop machines (Ubuntu 14.04 LTS, Intel i7-4770S with 4 cores at 3.10 GHz, 16GB RAM) that communicate over a 1Gbit/s LAN. To achieve long-term security according to NIST [BBB+07], we set the symmetric security level \(t\) and statistical security parameter \(\kappa\) to 128 bit if not stated otherwise. All results are aggregated over 30 independent runs and given by the mean and standard deviation (to improve readability, we leave out the standard deviation when it is below 1\% of the measurement).

4.4.8.1 Hyperplane Classifier

The fastest previous linear classifier is due to Bost et al. [BPTG15]. We obtained their code\(^3\) and evaluate Hyperplane against theirs on the following three datasets from the UCI Machine Learning Repository: Bost et al. choose the small Wisconsin Breast Cancer Diagnostic (WBCD) dataset\(^4\) with \(n = 32\) features and \(k = 2\) classes and the medium size Credit Approval (Credit) dataset\(^5\) with \(n = 48\) features and \(k = 2\) classes. We adopt this choice and additionally choose the large Human Activity Recognition (HAR) dataset\(^6\) with a total of \(n = 561\) features and \(k = 6\) classes to provide a greater variety.

Accuracy

We quantify the accuracy of Hyperplane by comparing against our own reference implementation of a Hyperplane classifier, which operates with double precision on plaintexts. Our goal is to show that Hyperplane only introduces negligible numerical errors compared to a traditional insecure implementation. The actual classification error then only depends on the trained classification model – no matter whether classification is executed securely or insecurely.

\(^3\)https://github.com/rbost/ciphermed (accessed: 2017-05-12)
4.4. Secure Classification Framework and Designs

We randomly select and classify 300 test vectors from each dataset and compare the numerical results as well as the predicted class labels. We observe a very low absolute numerical error of $2.46 \times 10^{-7}$ ($\sigma = 2.71 \times 10^{-7}$) between the results of the reference implementation and our Hyperplane when using a bitlength of $l = 64$ bit. For $l = 32$ bit, the error is slightly higher as less decimals can be represented with fixed point precision. More importantly, Hyperplane and the reference implementation version predict exactly the same target classes in all test cases for both choices of $l$.

We thus conclude that our secure Hyperplane protocol is practically as accurate as an insecure implementation.

**Runtime and Communication**

We measure runtime and communication overheads divided into onetime, precomputation, and online overheads. Onetime overheads are due to initialization, e.g., key generation; they may require knowledge of the classification model and of the protocol peer but must be independent of the number of subsequent classifications. Precomputation relates to all tasks that can be computed ahead of the concrete classification step, i.e., without knowing the feature vectors that have to be classified, but depending on the total number of classifications that will be carried out. The online phase comprises all overheads that cannot be shifted to the previous phases.

Additionally, we increase the Round Trip Time (RTT) of the underlying network from 1 ms to 40 ms up to 100 ms to single out the effects of the communication and round complexity of the evaluated approaches. We further configure a bitlength of $l = 64$ bit for high accuracy and use short-term security with $t = 80$ bit. Note that
Even though our parameter choice thus yields a very defensive comparison in favor of Bost et al.'s hyperplane classifier, our approach achieves significant performance improvements as detailed in the following.

Figures 4.6(a) to 4.6(c) compare the runtime and Figure 4.6(d) the communication overheads of Hyperplane (Our) and the Bost et al.'s hyperplane classifier [BPTG15] (Bost) on the chosen datasets for classifying a single feature vector. In fast networks (1ms RTT), our approach presents an $17.02 \times$ improvement (average over all datasets) which decreases to $1.38 \times$ and $0.91 \times$ in slower networks (40 ms and 100 ms RTT, respectively). Our approach is especially useful when classifications are performed only sporadically such that precomputations can be computed in between. In this case, we only need to perform the online phase which is faster by $21.01 \times$, $1.54 \times$, and $1.40 \times$, for the different network scenarios. In contrast, in high throughput scenarios, i.e., where huge batches of classifications have to be performed at the same time, the approach by Bost et al. is superior to ours in all but the 1ms network scenario due to its smaller combined precomputation and online overheads.

Our approach requires more communication on all three datasets than that of Bost et al., which explains its degrading performance in networks with higher RTT. However, much of Hyperplane's communication overheads can be shifted into the precomputation phase, which explains the high efficiency of our online phase. With overall communication of only a few MB even for the largest dataset, communication overheads are clearly feasible even in mobile scenarios.

4.4.8.2 Artificial Neural Networks

The most efficient secure ANN classifier is the Cryptonets approach by Dowlin et al. [DGBL16]. The ANN of [DGBL16] is designed to recognize handwritten digit numbers from the MNIST dataset\(^7\) and has the following five-layered structure: The first layer with $m_1 = 845$ neurons computes a convolution of the input image with a window size of $5 \times 5$ pixels, a stride of $2 \times 2$ pixels, and five different weighted sums for each window. On the second layer with $m_2 = m_1 = 845$ neurons, each output of the first layer is squared. The third layer with $m_3 = 100$ neurons pools the outputs of the previous layer by computing 100 weighted sums over all 845 outputs of the previous layer. On the fourth layer with $m_4 = m_3 = 100$ neurons, the output of each neuron on the previous pooling layer are squared again. The final layer with $m_5 = 10$ neurons computes 10 weighted sums over the outputs of the previous squaring layer. Each sum corresponds to one of the 10 digits that are the target classes.

Since we could neither obtain their code nor the trained networks, we exactly rebuild the described ANN model with the Lasagne\(^8\) neural networks library which is implemented in Python and trained our own networks on the MNIST dataset.

\(^7\)http://yann.lecun.com/exdb/mnist/ (accessed: 2017-05-12)
\(^8\)https://github.com/Lasagne/Lasagne (accessed: 2017-07-17)
4.4 Secure Classification Framework and Designs

Our Single classification

Runtime [s]

0 5 10 15

100 s

Batchsize

0 200 400 600 800 1000 1200 1400 1600

Figure 4.7 Runtime of ANN (left) and comparison to Dowlin et al. [DGBL+16] (right).

Accuracy

We quantify the accuracy of ANN by comparing against our insecure reference implementation which is based on Lasagne and operates with double precision. As before, our goal is to show that our secure ANN shows identical or sufficiently similar accuracy as a traditional insecure implementation such that the actual classification error then only depends on the trained classification model – no matter whether classification is executed securely or insecurely.

In 30 independent runs, we randomly select and classify 300 test images from the MNIST dataset and compare the numerical results as well as the predicted class labels. We observe an absolute numerical error of $8.60 \times 10^{-2}$ ($\sigma = 7.42 \times 10^{-2}$) between the results of the reference implementation and our secure ANN. Notably, this error is four to five orders of magnitude higher than for the previously evaluated Hyperplane. This difference is primarily due to the higher number of sequential multiplications performed in ANN (one on each of the five layers) than in Hyperplane (one for a single weighted sum). Indeed, we observe that the numerical errors of ANN grow with each layer. We thus expect that neural networks with more layers (as becoming popular in the context of Deep Learning) require improvements to our fixed point-precision number representation, in particular a rescale protocol with deterministic rounding.

Despite the perceivable numerical inaccuracies, our secure ANN classifier and the insecure reference implementation predict exactly the same target classes in all 300 test cases. We thus conclude that ANN is challenged by but still achieves sufficient accuracy for the non-trivial network proposed by Dowlin et al. [DGBL+16].

Runtime and Communication

Figure 4.7 (left) shows the performance of our secure ANN protocol for the described neural network model on the MNIST dataset. We observe that our approach has low latency and performs a single classification in the order of seconds, i.e., between 0.60 s ($\sigma = 0.01 s$) in the 1 ms setting to 12.08 s ($\sigma = 2.02 s$) in networks with 100 ms RTT. 88.27% of the runtime overheads in our approach are due to precomputing multiplication triples for the secure multiplication of additive shares; only 11.73% of
the runtime overheads fall into the critical online phase. The communication overheads amount to 0.02 MB, 74.13 MB, and 2.20 MB in the onetime, precomputation, and online phase, respectively.

The Cryptonets approach is completely different to ours as it optimizes for throughput. It is based on FHE with parameters chosen such that the protocol can encode batches of up to 8192 images in a single ciphertext and classify all of them together in an SIMD fashion. On a host with 3.5 GHz and 16 GB this requires a fixed 697 s for encryption, applying the network, and decrypting the result with an additional 106 ms per image in the batch and causes 595.5 MB of traffic. On the one hand, this approach is thus highly efficient for classifying huge batches of images. On the other hand, the significant fixed runtime and classification overheads are clearly disadvantageous when classifications are performed on small batches.

In Figure 4.7 (right), we compare the runtime of our approach against Cryptonets and denote the batch sizes for which both approaches have equal runtimes. Since Cryptonets is a two-rounds protocol, it is almost unaffected by network latency and therefore represented by single line in Figure 4.7. We observe that our approach is faster for batches of up to 1468, 146, and 59 classifications in the 1 ms, 40 ms, and 100 ms RTT network scenarios, respectively, and causes less communication overheads for batches below a size of 8. In conclusion, our approach trades throughput for one to three orders of magnitude (depending on the underlying network) lower classification latency for single classifications and thus represents a completely different point in the design space of secure ANN classifiers compared to the approach due to Dowlin et al. [DGBL+16].

4.4.8.3 Naive Bayes

We compare against fastest existing secure Naive Bayes classifier by Bost et al. [BPTG15]. Bost et al. evaluate their approach on the following three datasets from the UCI Machine Learning Repository which we also chose: i) the original version of the Wisconsin Breast Cancer (WBC) dataset\(^9\) with \(n = 9\) features and \(k = 2\) classes, ii) the Nursery dataset\(^10\) with \(n = 9\) features and \(k = 5\) classes, and iii) the standardized Audiology dataset\(^11\) with \(n = 70\) features and \(k = 24\) classes.

Accuracy

As before, we quantify the accuracy of NaiveBayes by comparing against a reference implementation operating with double precision on plaintexts with the goal to prove identical behavior. We randomly select and classify 300 test vectors from each dataset and compare the numerical results as well as the predicted class labels.

We observe a very low absolute numerical error of \(6.37 \times 10^{-8}\) \((\sigma = 6.70 \times 10^{-8})\) between the results of the reference implementation and our discrete NaiveBayes (using EvalProbMass to evaluate probability mass functions). For the continuous NaiveBayes (using Gaussian to sample the Normal distribution), the absolute

4.4. Secure Classification Framework and Designs

Figure 4.8 The runtime and communication overheads of NaiveBayes on three different datasets and network scenarios.

As before, we measure the runtime in the onetime, precomputation, and online phase, vary the RTT of the underlying network from 1ms, to 40ms, to 100ms, configure a bitlength of \( l = 64 \) bit for high accuracy, and set \( t = 80 \) bit for short-term security. Although this parameter choice yields a similarly defensive comparison to Bost et al. as made for Hyperplane before (cf. Section 4.4.8.1), our Naive Bayes classifier again clearly outperforms theirs.

Figures 4.8(a) to 4.8(c) compare the runtime and Figure 4.8(d) the communication overheads for classifying a single feature vector using our NaiveBayes on discrete features (Dis.), NaiveBayes with Gaussian features (Gau.), and Bost et al.’s approach. Regarding the total runtime, our NaiveBayes protocol on discrete features achieves notable improvements of 63.47×, 6.24×, and 4.99× averaged over all three datasets in the 1ms, 40ms, and 100ms RTT networks, respectively. NaiveBayes with an underlying Gaussian distribution is less efficient but still significantly improves upon Bost et al.’s approach by 32.44×, 3.41×, and 2.42×. Both our approaches thus have much lower latency for sporadic classifications.

numeical error of \( 4.26 \times 10^{-5} \) \((\sigma = 3.78 \times 10^{-5})\) is three orders of magnitude higher due to the use of multiplications and rescaling that introduce numerical inaccuracy additional to the limited fixed-point precision. For both the discrete and continuous version, NaiveBayes and the reference implementation predict exactly the same target classes in all test cases. We thus conclude that our secure NaiveBayes protocol is practically as accurate as an insecure implementation.
We observe that a large fraction of the overheads of the approach by Bost et al. is due to onetime overheads. These overheads would amortize when larger batches of classifications are performed at the same time. We thus compare runtimes of the three approaches without considering onetime overheads in order to compare the throughput of the three approaches. Our discrete NaiveBayes then achieves a \(37.24 \times\), \(4.11 \times\), and \(3.78 \times\) higher throughput than Bost et al. in the three respective network settings, while the less efficient Gaussian NaiveBayes still improves by \(13.40 \times\), \(2.05 \times\), and \(1.88 \times\). Our improvements are due to our highly efficient EvalProbMass primitive as well as the efficient combination of additive sharings with GCs in contrast to the costly HE primitives used by Bost et al. \[BPTG15\].

In terms of communication overheads, only the discrete NaiveBayes compares favorably while the Gaussian NaiveBayes has a significantly higher communication complexity (most of which can be precomputed). Still, Gaussian NaiveBayes has the overall most efficient online phase (in terms of both runtime and communication) which is important for scenarios that require low latency for sporadic classifications.

### 4.4.8.4 Forward and Viterbi

For our evaluation of Forward and Viterbi, we create a synthetic dataset of ergodic (fully connected) HMMs to serve as inputs. Ergodic HMMs are the most general case and also the most challenging in terms of performance and accuracy (especially for STC but also on plaintexts) as the set of predecessors during the recursion steps of the Forward and Viterbi algorithm is the entire state space of the HMM. We consider real-world HMMs with special architectures that allow optimizations in the evaluation of our bioinformatics and localization use cases (Sections 4.4.9.2 and 4.4.9.3).

Our dataset consists of random and circular HMMs as well as matching and non-matching observation sequences. Random HMMs and non-matching sequences are sampled completely at random, while for the circular HMMs, we sample state transitions and emission probabilities from a Gaussian distribution centered on the topologically next state leading to a roughly circular structure. With the intention to create good matches, we sample observation sequences for the circular HMMs as a noisy linear walk through the states of the HMM.

Both datasets contain HMMs with a varying number of states \(N = 10, 20, \ldots, 100\), emission alphabet sizes \(M = 10^2, \ldots, 10^4\), and observation sequences with length \(T = 10, 20, \ldots, 100\). This selection of parameters covers a wide range of actual use cases and choices made in related work \[Rab89, PRSR11, PRRS13, AB13\].

#### Accuracy

Our approach to secure Forward and Viterbi computation introduces numerical inaccuracies at two points, i) through the fixed-point representation of probabilities and rescaling with probabilistic rounding error (Section 4.4.1), and ii) through the approximation of logsum operations (Section 4.4.5). In the following, we quantify the achieved accuracy of our secure Forward and Viterbi by comparing against a reference implementation on plaintexts, i.e., the widely-used Natural Language Toolkit \[BKL09, nlt17\], which is implemented in double precision in Python.
Figure 4.9 Relative error of \texttt{Forward} for PLA sizes $k \in \{2, 4, \ldots, 32\}$ and \texttt{Viterbi} (Vit.) with varying bitlengths $l = 32\text{ bit}$ (left) and $l = 64\text{ bit}$ (right).

Figure 4.9 plots the errors of \texttt{Forward} and \texttt{Viterbi} relative to the reference results. On the x-axis, we vary the number of approximation intervals $k \in \{2, 4, \ldots, 32\}$ of the PLA (only for \texttt{Forward}); left and right plots vary the bitlength $l \in \{32, 64\}$ (a higher bitlength allows more decimals in our fixed-point number representation and hence more numerical accuracy as detailed in Section 4.4.1). We show the second and third quartiles (boxes), the min and max (whiskers), the mean (stars), and median (middle of boxes) of the relative errors.

As expected from our detailed evaluation of \texttt{LogSum} (Appendix A.4), the error of \texttt{Forward} decreases as we increase the accuracy of the PLA by increasing the number of approximation intervals $k$. While an approximation with only $k = 2$ intervals causes significant errors, the error decreases quickly for bigger values of $k$. Already, $k = 4$ achieves a mean error below 0.3\% for both bitlengths $l$ while setting $k = 8$ reduces even the maximum error to at most 0.1\%. We observe that the average accuracy does not significantly improve beyond $k = 8$. This is due to the numerical errors introduced by the fixed-point number representation and rescaling protocol that are independent of $k$. Interestingly, using a higher bitlength of $l = 64\text{ bit}$ shows only marginal improvements to the average accuracy. However, the variance of the error is reduced and outliers are less extreme.

Figure 4.9 also shows that \texttt{Viterbi} achieves higher accuracy compared to \texttt{Forward}, i.e., even outliers are well below 0.01\% for $l = 32\text{ bit}$ and even below 0.0001\% for $l = 64\text{ bit}$. This is expected as we replace the slightly inaccurate \texttt{LogSum} with \texttt{Max\ArgMax} which does not introduce any errors. Additionally, the logspace variant of the Viterbi algorithm involves no multiplications which eliminates a second source of inaccuracies, the \texttt{Rescale} protocol. Thus, the only source of errors in \texttt{Viterbi} is due to the initial representation of probabilities as fixed point logarithmic values. This error is very low since we can use most of the available $l$ bits to represent decimals when only dealing with additions. This also explains the orders of magnitude improvement in accuracy when increasing bitlength to $l = 64\text{ bit}$.

In conclusion, our results confirm the expected: \texttt{Viterbi} is already very accurate for $l = 32\text{ bit}$ and even achieves zero error in most cases for $l = 64\text{ bit}$; similarly, more approximation intervals $k$ and higher bitlength $l$ improve the numerical accuracy of \texttt{Forward}. Unfortunately, an increase of either parameter decreases the overall...
4. Privacy-preserving Classification and Pattern Recognition

performance of \textsc{Forward} and \textsc{Viterbi} as we see in the next section. In practice, tuning parameters \(l\) and \(k\) thus allows us to strike a trade-off between accuracy and performance – the real-world use cases presented in Sections 4.4.9.2 and 4.4.9.3 put this trade-off into perspective.

\textbf{Runtime and Communication}

We first analytically derive the critical parameters for the runtime of \textsc{Forward} and \textsc{Viterbi} and then thoroughly evaluate their performance. The runtime complexity of the original Forward and Viterbi algorithms is \(O(TN^2)\), i.e., quadratic in the number of states \(N\) and linear in the number of observations \(T\). In the secure \textsc{Forward} and \textsc{Viterbi} protocol, we additionally need to share all emission scores \(\hat{b}_i(o)\) securely using \textsc{EvalProb} which scales in \(O(TM)\) with \(M\) the size of the emission alphabet. Thus, \(N\), \(T\), and \(M\) are the critical parameters for which we have to analyze the runtime and communication overheads.

\textbf{Number of States} \(N\). Figure 4.10 plots the runtime of \textsc{Forward} and \textsc{Viterbi} on HMMs with an increasing number of states \(N\) for different PLA sizes \(k\) and bitlengths \(l\) (with fixed \(M = 1000\) and \(T = 10\)). As indicated by the overall complexity of the Forward and Viterbi algorithm, the runtime increases quadratically in the number of states. We observe qualitatively the same growth for the communication overhead of both \textsc{Forward} and \textsc{Viterbi}. Still, both approaches perform reasonably well, e.g., \textsc{Forward} on a fully connected HMM with \(N = 100\) states configured for 99\% accuracy (\(k = 4, l = 32\) bit) requires 56.02 s and 1.59 GB of communication while \textsc{Viterbi} is roughly 3\times more efficient, requiring only 18.25 s and 0.56 GB.

Increasing the PLA size \(k\) or bitlength \(l\) also increases runtime and communication of \textsc{Forward} linearly. This is due to the fact that the runtime is dominated by the \((T-1)N(N-1)\) invocations of \textsc{LogSum} whose core is a GC of size linear in \(k\) and \(l\). In total, the \textsc{LogSum} calls account for 98.51\% of the overall runtime. The rest of the computational overhead is due to \textsc{EvalProbMass}, while the overhead of the operations on additive sharings are negligible.

In comparison, \textsc{Viterbi} is faster than \textsc{Forward} by a factor of 2.25\times (\(k = 2, l = 32\) bit) to 12.34\times (\(k = 32, l = 64\) bit) on an HMM with 100 states 10,000 emissions

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{Figure_4.10.pdf}
\caption{Runtime of \textsc{Forward} (colored lines) and \textsc{Viterbi} (gray line) on HMMs with a different number of states \(N\) (x-axis) for PLA size \(k\) (lines) and bitlength \(l\) (left and right).}
\end{figure}
and 100 observations. This is because \textsc{MaxArgmax} scales only in \( l \) and is much more efficient than \textsc{LogSum} as the involved GCs are much smaller. In total, the calls to \textsc{MaxArgmax} account for 95.35\% of the overall runtime of \textsc{Viterbi}. Again, the rest of the overheads are due to \textsc{EvalProbMass}.

As expected for STC, the communication overheads of both \textsc{Forward} and \textsc{Viterbi} are significant and may overtax especially mobile users. We explain in Section 4.5.2 how \textsc{Forward} and \textsc{Viterbi} can be securely outsourced to an untrusted computation cloud to cater for constrained mobile devices.

\textbf{Length of the Observation Sequence} \( T \). Figure 4.11 plots the runtime of \textsc{Forward} and \textsc{Viterbi} for an increasing number of observations \( T \) for different PLA sizes \( k \) and bitlengths \( l \) (with fixed \( N = 10 \) and \( M = 1000 \)). Runtime and communication scale linearly in \( T \) as the number of required \textsc{LogSum} and \textsc{MaxArgmax} operations scales linearly in \( T \). For example, processing \( T = 10 \) observations in \textsc{Forward} \((l = 32\text{bit}, k = 4)\) costs 0.64 s \((\sigma = 0.01\text{s})\) and 15.45 MB which grows by roughly \( 10\times \) to 6.84 s \((\sigma = 0.11\text{s})\) and 167.05 MB for \( T = 100 \) observations. In comparison, \textsc{Viterbi} is again roughly \( 3\times \) more efficient and requires only 0.22 s \((\sigma = 0.00\text{s})\) and 0.35 MB to 2.44 s \((\sigma = 0.04\text{s})\) and 68.12 MB.

\textbf{Size of the Emission Alphabet} \( M \). The alphabet size only influences the performance of \textsc{EvalProbMass} which accounts for less than 2\% and 5\% of the total overheads of \textsc{Forward} and \textsc{Viterbi}, respectively. Due to its small impact, we refer to the dedicated evaluation of the \textsc{EvalProbMass} primitive in Appendix A.3 and omit further measurements here.

\textbf{Comparison to Related Work}

We compare the performance of \textsc{Forward} and \textsc{Viterbi} quantitatively to those related works that we have already discussed qualitatively in Section 4.2.2. Since we could not obtain the source code from the authors, we resort to the measurements provided in the corresponding papers. Thus, the presented measurements were obtained on different machines and can only serve as a rough indication, although we make the same or more defensive choices for all other evaluation parameters.
Pathak et al. [PRSR11, PRRS13] report a runtime of 784.58 s for the evaluation of one HMM with $N = 5$ states on an observation sequence of length $T = 98$ on a 3.2 GHz CPU. For a comparison, we restricted SHIELD to run on one core at 3.1 GHz and set the PLA size to $k = 4$ and symmetric security to $t = 80$ bit which achieves comparable accuracy and security as in [PRSR11, PRRS13]. In this setting, we measure the runtime of FORWARD at 6.21 s which is 126x faster. Although these results can only serve as a rough indication (being measured on different hosts), they still demonstrate our notable performance improvements.

Aliasgari et al. [AB13, ABZS13], Kamm et al. [KW15], and Demmler et al. [DDK+15] propose secure computation on floats which could be used to implement a secure Forward or Viterbi algorithm over probability space and using normalization [Rab89] to avoid underflows. Then, only selection of the required emission probabilities is not straightforward, but can be realized through component-wise multiplication which is the fastest previous method proposed in [FDH+11]. To draw a concrete comparison, we estimate runtimes by counting the calls to the required primitives and weighting them according to the performance measurements presented in [DDK+15], for which we choose a high batch size of 1000 which yields very defensive estimates (i.e., in favor of the related works). In this setting, we estimate that Forward computation of $T = 100$ observations over an HMM with $N = 10$ states and an alphabet of $M = 100$ symbols would cost at least 251.82 s, 51.76 s, and 39.37 s using the primitives from [ABZS13], [KW15], and [DDK+15], respectively. In contrast, even when parametrized with $k = 8$ for high accuracy, FORWARD requires only 9.85 s which is approximately 25×, 5×, and 4× faster.

Aliasgari et al. [ABB16] run experiments for secure Viterbi computation in a comparable setup to ours, i.e., between two hosts with 3.2 GHz connected over a 1 Gbit/s LAN, but provide only short to medium security levels (80 bit to 112 bit of symmetric security as opposed to 128 bit in our setup). With more than 400 min for securely computing the Viterbi algorithm on an HMM with only $N = 6$ and an observation sequence of length $T = 96$, the reported runtimes in the two-party setting for the semi-honest model are clearly prohibitive; the runtime of 23 s in the multi-party setting is much more reasonable. Our secure VITERBI protocol requires only 0.19 s on an HMM with $N = 10$ states and $T = 100$ observations which is an improvement by at least five orders of magnitude compared to the two-party setting and almost two orders of magnitude compared to the multi-party setting – since we use a larger HMM and higher security level, this is a very defensive comparison.

### 4.4.9 Use cases

Having developed a representative range of secure classifiers and evaluated their performance on different datasets, we now demonstrate real-world use cases for three of them: First, we build a privacy-preserving spam filter upon NAIVEBAYES that allows a service provider to obliviously classify a user’s messages without revealing the spam model or learning the messages. Second, we integrate FORWARD and VITERBI into a widely established bioinformatics framework to realize privacy-preserving sequence alignments for different bioinformatics services such as genetic disease testing. Third, we present a privacy-preserving indoor localization system that uses VITERBI to protect the user’s position and the service provider’s localization models.
Realizing these different use cases allows us to validate important aspects of our designs: First, each use case brings new individual requirements that require adoption or enable specific optimizations – this allows us to validate the flexibility, extensibility, and composability of our designs. Second, real-world use cases provide us with realistic problem instances – testing our approaches on them proves their scalability and complements our general performance evaluation. Finally, each use case touches on an existing or emerging digital privacy issue – this emphasizes the importance and timeliness of our work.

4.4.9.1 Spam Filtering

Spam filtering can be implemented in a straightforward manner using a multinomial Naive Bayes classifier [APK+00]. However, this simple approach causes two problems: First, many words will occur only rarely in a message [Sic75] which makes it hard to determine the priors of these words. Second, messages can contain both words that strongly indicate spam and words that strongly indicate non-spam (ham).

To solve these issues, modern approaches build on the Chi-squared distribution [Rob03]. Training a spam model using this approach is similar to the training of a conventional Naive Bayes classifier. We analyze emails labeled spam or ham and count for each word \( w \) how many times it occurs in a spam message and ham, denoted \( \#_S(w) \) and \( \#_H(w) \), respectively, with \( \#_S \) and \( \#_H \) the total number of spam and ham messages. We then compute ratios \( s(w) = \frac{\#_S(w)}{\#_S} \) and \( h(w) = \frac{\#_H(w)}{\#_H} \) which correspond to the likelihoods \( p(w|\text{Spam}) \) and \( p(w|\text{Ham}) \). Instead of the posteriors \( P(\text{Spam}|w) \), we calculate beliefs \( f(w) \) which better account for rare words [Rob03]:

\[
f(w) = \frac{(s \cdot x) + (n \cdot p(w))}{s + n}
\]  

with \( p(w) = \frac{\#(w)}{n} \), \( n = \#_S(w) + \#_H(w) \) the total number of messages, \( x \) the estimated probability that an unknown word is first encountered in a spam message, and \( s \) a weight parameter (\( x \) and \( s \) are chosen by the individual spam filter implementation).

From the belief scores \( f(w) \) for individual words \( w \), we then use the inverse Chi-squared distribution to compute an overall score for how desired the message is (we can call it the *haminess*):

\[
H = C^{-1} \left( -2 \ln \left( \prod_w f(w) \right) \cdot 2n \right)
\]  

Analogously, a score \( S \) is calculated using beliefs \( 1 - f(w) \) for how undesired a message is (the *spaminess*). Finally, ham and spam are interpreted as independent classes and the spam filter determines the more likely class according to the following indicator which approaches one for ham and zero for spam messages:

\[
I = \frac{1 + H - S}{2}
\]  

Realizing this approach in a secure protocol, numerical stability becomes a challenge once again: Messages can contain a large number of words and for most of them
one of the beliefs \( f(w) \) or \( 1 - f(w) \) becomes very small and may cause underflows. We thus choose a logspace representation. Furthermore, the calculation of \( H \) and \( S \) requires computing the multivariate inverse chi-squared function to which our FUNCTIONAPPROX protocol does not apply. To solve this issue, we compute \( H' = \sum_i \log(f(w_i)) \) and \( S' = \sum_i \log(1 - f(w_i)) \) securely, then the user recombines \( H' \) and \( S' \) and determines the indicator \( I \) locally by

\[
I = \frac{1 + C^{-1} (-2H', 2n) - C^{-1} (-2S', 2n)}{2} \quad (4.30)
\]

Extending our discrete NaiveBayes (Protocol 4.7 in Section 4.4.4) to this approach is straightforward: \( U \) and \( S \) use EvalProbMass to obliviously share the belief scores \( f(w) \) and \( 1 - f(w) \), compute shares \( \langle H' \rangle \) and \( \langle S' \rangle \) locally, and open the result to \( U \). We denote this approach by \text{AsSpam} (as it is based on additive shares).

For a comparison, we realize an alternative approach, \text{HeSpam}, in which we compute \( H' \) and \( S' \) using \text{HE} according to the design of the Naive Bayes classifier due to Bost et al. [BPTG15]: The service provider precomputes and encrypts all belief scores \( f(w) \) and \( 1 - f(w) \) for each word \( w \) in the spam model using an additive \text{HE} scheme and sends all encryptions to the user. \( U \) can then compute the scores \( H' \) and \( S' \) locally under encryption and needs only a single interaction with \( S \) to obtain the decrypted scores (using additive blindings such that \( S \) does not learn the result).

**Accuracy**

We evaluate the accuracy of \text{AsSpam} and \text{HeSpam} on the CSDMC2010 SPAM corpus.\(^{12}\) This dataset contains a total of 8619 messages of which 2949 are labeled ham, 1378 as spam, and 4292 messages are unlabeled. We employ the widely used Bogofilter\(^{13}\) to train a spam model using all labeled emails, leading to a model over 146,515 known words.

Using the trained model, we classify 300 spam messages as well as 300 ham messages using \text{AsSpam} and \text{HeSpam} where the classification of Bogofilter as taken as reference. Both approaches achieve very high numerical accuracy: The relative error is \( 2.36 \times 10^{-3} \) (\( \sigma = 5.96 \times 10^{-2} \)) for \text{AsSpam} while \text{HeSpam} is even slightly more accurate. The categorical classification result (i.e., spam or ham) of both \text{AsSpam} and \text{HeSpam} is exactly equal to that of Bogofilter due to the high numerical accuracy.

**Runtime and Communication**

Depending on the network’s latency, \text{AsSpam} requires a runtime of 48.33ms in the 1 ms RTT to 1514.90ms in the 100 ms RTT network scenario as well as a total of 7.62 MB of communication to classify a single message. Note that in this approach, \( U \) and \( S \) need to run EvalProbMass for each word in the model to not disclose which words are contained in the message. The runtime of \text{AsSpam} is thus independent of the length of the message and only depends on the size of the spam filter model.


In contrast, our second approach, $H$SpAM, requires only two rounds of communication and thus the network’s RTT has only negligible impact. Not considering RTT for simplicity, $H$SpAM runs for 0.21 ms to 5.26 ms on the shortest and longest message in the dataset, respectively (and 2.64 ms ($\sigma = 1.97$ ms) in average over all messages). However, $H$SpAM incurs significant onetime runtime and communications costs of 180.43 s ($\sigma = 2.77$ s) and 76.77 MB ($\sigma = 0.00$ MB) for the initial encryption of the spam model and transferring it to the user.

Figure 4.4.9.1 compares both approaches against each other to determine after how many classifications the significant onetime costs amortize and make $H$SpAM more efficient than AsSpAM. Since $H$SpAM has only one round of communication, we plot only the performance for the 1 ms RTT network – performance in the 40 ms and 100 ms setting increases only by this RTT which is negligible compared to the overall runtime. We observe that $H$SpAM outperforms AsSpAM for batches of 3949, 296, and 119 classifications or more in the three network scenarios. With regards to communication, $H$SpAM outperforms AsSpAM already after 10 classifications.

In conclusion, our two secure spam filters have vastly different performance characteristics: AsSpAM has almost no setup costs but a slower online phase – it is practical for sporadically classifying messages with low latency. $H$SpAM, in contrast, has high setup costs but scores messages very efficiently afterwards – it provides high throughput when spam models are fixed and sufficiently many messages have to be classified.

### 4.4.9.2 Bioinformatics Service

Recent advances have made whole genome sequencing fast, accurate, and affordable for the masses. It is widely expected that whole genome sequencing will pave the way for innovative research and novel applications [DCFGT12, ZSG14]. As we can already observe, an industry emerges that offers genomic services such as drug testing or diagnosis of diseases based on proprietary research [ZSG14]. To remain competitive, service providers will need to protect the statistical models upon which their businesses are built. On the other hand, users of such services are nowadays required to share their genomic data which is most sensitive information [ACHT13]. Our approaches to secure computations on HMMs allow preserving the service provider’s intellectual property while offering strong protection for users’ genomic data.
To present a concrete use case, we consider the following genetic disease testing scenario: The service provider holds a set of HMMs that model specific diseases. The user holds an observation sequence, e.g., parts of her sequenced genome, that she wants to test against the service provider’s database. Concretely, we use HMMs from the Pfam database [pfa15], which contains 16,295 protein families that relate to certain phenotypes and diseases, among others. HMMER [hmm] is a well established tool in the bioinformatics community and is used to query the Pfam database. We thus implemented Forward and Viterbi in the most recent HMMER version14.

It is important to note that HMMER and the HMMs in the Pfam database use the profile HMM architecture. Profile HMMs have a special structure that limits the number of transitions which significantly speeds up Forward and Viterbi computation as less predecessors have to be considered during the recursion step. Since we adapt these optimizations which are specific to the profile HMM architecture, the results presented in this section are not comparable to those presented in Section 4.4.8.4 which were obtained on generic fully connected HMMs. It becomes clear, though, that our Forward and Viterbi protocols are flexible enough to capitalize on the optimization potential offered by the reduced profile HMM architecture.

From Pfam [pfa15], we choose the same models as in [FDH11], i.e., SH3_1 (Length $L = 48$), Ras ($L = 162$), BID ($L = 191$), and added one of the smallest models from Pfam, Extensin_1 ($L = 10$), another medium size model, RibosomalS3C ($L = 83$) as well as two of the largest models, IDO ($L = 408$) and 3HBOH ($L = 689$). Here, $L$ denotes the number of nodes in the profile HMM where each node has three distinct states; together with four special states, a profile HMM thus has a total of $N = 3L + 4$ states. The average length of HMMs in Pfam is 175 and more than 98.5% of the HMMs have a length smaller than 3HBOH, the largest model we consider.

Observation sequences typically have the same length as the HMM ($T = L$) and we choose two types: i) matching sequences where we use the seeds on which the respective models were trained and ii) non-matching sequences which we choose randomly from the seeds of other models. The considered profile HMMs are built over the amino acids alphabet which has $M = 20$ symbols (as opposed to those over the RNA alphabet which has only $M = 4$ symbols).


Figure 4.13 Relative error of Forward (for PLA sizes $k \in \{2, 4, 8\}$) and Viterbi ($V$) on real-world profile HMMs (groups) from the Pfam database [pfa15].
4.4. Secure Classification Framework and Designs

Accuracy

Figure 4.13 plots the relative error Forward and Viterbi introduce in comparison to the real scores computed by the HMMER framework with double precision on plaintexts. We restrict the evaluation to $k \in \{2, 4, 8\}$ (for Forward) and $l = 32$ bit since these choices achieve the best trade-off between accuracy and performance according to the previous results for ergodic HMMs (Section 4.4.8.4). A small PLA with only $k = 2$ approximation intervals leads to large errors of Forward that grow roughly linearly with the combined length of the model and observation sequence. Increasing to $k = 4$ and $k = 8$, the error of Forward mostly drops below 1%. For Viterbi, the error is always well below 0.1%. In the latter two cases, the error has no apparent correlation to the length of model and observation sequence anymore.

Considering our use case, the more important question is whether our Forward and Viterbi are accurate enough to distinguish matching from non-matching sequences. To answer this question, we classify sequences according to the noise cutoffs (NC) and trusted cutoffs (TC) specified for each model in the Pfam database: Anything below the NC can safely be considered a non-matching sequence and anything above the TC a match. Both Viterbi and Forward (for $k \in \{4, 8\}$) are able to perfectly distinguish between matching and non-matching sequences. Notably, even for $k = 2$ classification is perfectly accurate for all but the largest model, 3HBOH.

Runtime and Communication

Figures 4.14 and 4.15 plot the performance and communication overhead for the chosen HMMs from the Pfam database. The x-axis denotes both length $L$ of the model and length $T$ of the observation sequence. Note that we disabled backtracking in Viterbi since in this use case we are only interested in the score of the observation sequence and not in the corresponding sequence of internal states. Disabling backtracking saves roughly one third of Viterbi’s overheads. The runtime and communication overheads are dominated by the invocations of LogSum and Max in Forward and Viterbi, respectively. Since the Forward and Viterbi algorithms in HMMER 3.1 require only $T(7L + 2)$ LogSum or Max operations (due to the reduced profile HMM architecture), runtime and communication grow linearly in both
Figure 4.15 Communication of Forward and Viterbi on profile HMMs from Pfam [pfam15] with different length $L$ on observation sequences of length $T = L$ (x-axis).

$t$ and $L$. The growth in Figures 4.14 and 4.15 is quadratic only because we increase both $L$ and $T$ (as observation sequences match the length of HMMs in this use case).

Using Forward, the smaller models can be computed in the order of seconds, e.g., 13.36s ($\sigma = 0.29s$) for SH3_1, while the larger models range in the order of minutes, e.g., 14.29min ($\sigma = 0.81\min$) for IDO. With Viterbi, the same models require only 1.80s ($\sigma = 0.05s$) and 2.37min ($\sigma = 0.07\min$). Similar to the runtime, the communication overhead is dominated by the calls to LogSum and MaxArgmax and thus grows quadratically as well. For Forward, we measure 10.36MB ($\sigma = 0.95\MB$) for the smallest model, Extensin_1, and up to 47.18GB ($\sigma = 0.59\GB$) for the largest model, 3HBOH. Using Viterbi, the communication overheads decrease to 2.10MB ($\sigma = 0.25\MB$) and 9.80GB ($\sigma = 0.17\GB$) for the same models.

Notably, Viterbi is almost one order of magnitude faster than Forward and reduces communication overheads by roughly $5 \times$, which is due to Max being much more efficient than LogSum. Since both algorithms classify sequences in exactly the same way (the computed numerical scores are very close), they are interchangeable in this use case. However, we emphasize that this is not always the case, e.g., Forward computation is an integral part for training HMM where it cannot be replaced by the Viterbi algorithm.

Finally, we note that the measured runtime overheads are reasonable even for the largest models. The communication overheads, in contrast, are likely to cause prohibitive costs or significant increases of the runtime when hosts are connected over networks where bandwidth is limited and traffic is at a premium, e.g., as typical for cellular networks. This makes evident the necessity to cater secure protocols to mobile deployments and, in general, to constrained deployments – we develop corresponding approaches in Section 4.5.

Comparison to Related Work

Franz et al. [FDH+11] evaluate their approach on profile HMMs as in our bioinformatics use case. Unfortunately, we could not obtain their source code and a direct comparison with our results presented in Section 4.4.9.2 is unreasonable for different reasons: First, the Forward and Viterbi algorithm in HMMER version 3.1 requires to
compute an additional $T \cdot L + 1 \logsum$ or max operations compared to version 2.3.2 used in [FDH+11]. Switching to version 2.3.2 reduces the overheads of our approach by approximately 14%. Second, Franz et al. do not implement any networking, yet we observe networking to cause non-negligible overheads even on the local loopback interface. Third, Franz et al. use short-term security levels while we use long-term security which causes additional overheads in comparison. Finally, the evaluation machines differ between a processor with 8 cores at 2.1 GHz used in [FDH+11] and a processor with 4 cores at 3.1 GHz machine used in our evaluation. Both implementations parallelize to multiple cores.

For a fairer but still defensive comparison, we run Forward and Viterbi over the loopback interface, configure short-term security levels, and switch to the HMMER 2.3.2 style of Forward and Viterbi computation. Franz et al. [FDH+11] report runtimes of 33s, 499s, and 632s for Forward computation on the Pfam models SH3_{-1}, Ras, and BID, respectively. In comparison, Forward requires only 8.15s, 89.92s, and 125.25s which is an $4.05 \times$, $5.55 \times$, and $5.05 \times$ improvement over [FDH+11], respectively. In the same setting and for the same models, the main author of [FDH+11] reports in [Fra11, Sect. 8.5.2.1] runtimes for Viterbi computation of 94s, 933s, and 1357s. In contrast, our Viterbi requires only 1.28s ($\sigma = 0.01s$), 14.59s ($\sigma = 0.05s$), and 19.76s ($\sigma = 0.05s$) which is an improvement of nearly two orders of magnitude (73.70$\times$, 63.95$\times$, and 68.66$\times$). However, we emphasize again that latter results were obtained on different machines and can thus serve only as a rough indicator.

**4.4.9.3 Use case: Localization**

WiFi-based localization and navigation systems are increasingly considered for indoor environments [VLW+12, LYS+14]. In these systems, a user’s location is determined by repeatedly matching supplied received signal strength indicator (RSSI) measurements against a signal propagation model of the environment. However, this enables the service provider to track users which gives rise to serious privacy concerns. As we show in the following, the user’s concern for location privacy and the business interests of the system provider in protecting mobility and signal propagation models can be reconciled using our Viterbi. In particular, we enable the service provider to localize the user without learning the user’s location and without the user learning the service’s localization model.
Figure 4.16 provides an overview of our privacy-preserving variant of the indoor localization system proposed in [VLW+12]. The system of [VLW+12] models a user’s unknown sequence of positions as the hidden internal states of an HMM learned from combing a signal propagation model, floor plan, and human mobility patterns. The Viterbi algorithm is then used to determine the user’s most likely path based on the RSSI measurements she supplies continuously while walking around the building. The authors show that the algorithm achieves higher localization accuracy than simple nearest-neighbor methods for which privacy-preserving variants have been discussed [RB13]. Thus, we aim to redesign the HMM-based approach proposed in [VLW+12] using our secure Viterbi in order to reconcile the user’s location privacy with the business interests of the service provider (to protect his sophisticated localization models). To this end, we need to adapt our initial design in two ways: First, we need to handle continuous emission probabilities, and, second, we need to provide live location updates to the user in a privacy preserving manner.

**Computing emission probabilities.** Our initial design of Viterbi takes into account only discrete probability distributions for state transitions and emissions (modeled as the state transition probability matrix $A \in \mathbb{R}^{n \times n}$ and the emission probability matrix $B \in \mathbb{R}^{n \times m}$ in Section 4.2). However, in the case of WiFi-based indoor localization, emission probabilities are typically continuous, i.e., $b_i(o)$ is modeled as a product of Gaussians centered around the mean RSSI of access point $d$ at state $s_i$ [VLW+12, Section IV.A]. To handle continuous distributions in Viterbi, we can create a representative subsample and then use EvalProbMass as before. EvalProbMass can efficiently handle thousands of subsamples which already provides reasonable accuracy for the localization use case at hand.

A second option is to compute the emission probabilities directly, which is both faster and more accurate. To this end, we transform the multi-variate Gaussians into logspace and drop all constants since they do not matter during the max and argmax computations that determine the recursion steps of the Viterbi algorithm. We are then left with the following simple equation for the emission probabilities

$$b_i(o) = \sum_{d=1}^{D} \frac{(o_{t,d} - \mu_{s_i,d})^2}{\sigma_{s_i,d}^2}$$

where $o_{t,d}$ is the RSSI measured between the user’s device and the $d^{th}$ access point at time step $t$ known only to the user while $\mu_{s_i,d}$ and $\sigma_{s_i,d}$ are the mean RSSI and standard deviation of the $d^{th}$ access point in state $s_i$ and only known to the service provider. Equation 4.31 thus depends on private inputs of the user and the service provider and must be computed securely using our Gaussian protocol.

**Updating the user’s position.** For navigation purposes, the user’s position must be updated continuously. This requires the user and service provider to repeatedly execute the termination steps of Viterbi (Protocol 4.12). Note that at each update interval, the user learns the most probable position given only her observation so far, $o_1, o_2, ..., o_t$. These positions may differ from the final Viterbi path and cause jumps during the localization and navigation process. We emphasize that this is inherent to the HMM-based localization approach of [VLW+12] and not due to the use of STC.

With these two modifications in place, we implemented a prototype of Viol et al.’s indoor localization system [VLW+12]. In the following, we briefly evaluate the overheads of our approach and then discuss the localization accuracy it achieves.
Runtime and Communication

The overall complexity for computing one location update is $O(N\prime N + ND)$ where $N\prime$ is the number of predecessors per state and $D$ is the number of access points considered during localization. We achieve $N\prime \ll N$ by using a realistic human movement model, e.g., no transition through walls, no long jumps, no movement at high speeds. Note that the user learns the structure of the HMM, i.e., which state has which predecessors, during the Viterbi protocol. We argue that this knowledge is insensitive since it corresponds to evident facts (such as the location of walls) that the user can simply observe by walking around the building. If this should be prevented nevertheless, all possible states $S$ must be considered as predecessors during the recursion step which increases complexity to $O(ND)$ per location update.

In Figure 4.17, we measure the average overheads of one location update for different number of states $N$. Based on the results of [VLW+12] and their actual HMMs, we defensively fix the number of predecessors to $N\prime = 0.1N$ and vary the number $D = 5, 20$ of access points. We observe that a fresh position can be computed in roughly 5s when the indoor space is divided into approximately 600 states. If we tolerate an update delay of 10s, we can even handle spaces with up to 900 states. Using more access points, e.g., to improve localization accuracy, has only a very minor impact on the performance (i.e., the lines in Figure 4.17 for $D = 5$ and $D = 20$ almost completely overlap) since computing the emission scores using Gaussian is very efficient and the performance overheads are dominated by the invocations of MaxArgmax which do not depend on $D$.

The communication overheads are mostly due to MaxArgmax and range in the order of tens to hundreds MBs per location update which is significant. This motivates the need for outsourcing schemes and bandwidth optimizations as we propose in the second main part of this chapter (Section 4.5).

Accuracy

Compared to the original system [VLW+12], our approach introduces numerical inaccuracies at two points, i.e., through the quantization of probabilities as fixed-point logspace values and during the computation of emission scores. However, these
errors propagate only additively in the logspace Viterbi and our experiments show that they can be neglected in the context of this use case as they do not decrease the localization accuracy, the primary quality metric in this use case.

Instead, the actual achievable localization accuracy (i.e., not the numerical accuracy) is limited by the maximum size of an HMM that Viterbi can decode within one location update interval. The original system [VLW+12], e.g., separates the space into $0.2m \times 0.2m \times 0.2m$ voxels amounting to thousands of states in their test setup. In contrast, our modified Viterbi with Gaussian emission probabilities can handle within a reasonable delay for navigation, say 10s, only approximately 900 states. Thus, the indoor space must be divided into coarser voxels which inevitably reduces the achievable localization accuracy (despite the high numerical accuracy of our approach). Still, this number of states is enough to achieve accuracy in the order of meters in medium to large size buildings, e.g., for room-level localization.

4.4.10 Summary and Future Work

We have developed the SHIELD framework comprising several efficient secure building blocks and showed how to compose them to secure Hyperplane, ANN, and Naive Bayes classifiers as well as to realize more complex pattern recognition tasks such as Forward and Viterbi computation on HMMs. The principal challenges we faced are balancing performance and numerical accuracy while providing security in the semi-honest model. Our approach to overcome these challenges are improved or novel hybrid protocol designs that efficiently combine three established STC techniques, i.e., arithmetic sharings, GCs, and OT. A thorough evaluation on different established datasets from the machine learning community and in different network settings shows that our secure classifiers outperform state of the art approaches in most cases and provide tunable numerical accuracy. We showcased our secure classifiers in three concrete use cases to validate scalability and accuracy on real-world problem instances as well as to test the flexibility and adaptability of our designs when faced with use case specific requirements and optimization potential.

We conclude this section with a short discussion of future work. We point out extensions and use cases that we deem directly implementable using the SHIELD framework as well as those that require more functionality and optimizations.

Further classification algorithms: We exemplarily realized five classification algorithms on top of the proposed secure building blocks that are representative for linear and Bayesian classification as well as Markovian processes – other algorithms in these classes of statistical methods can be realized directly using our methods, e.g., SVMs, perceptrons, logistic regression, multinomial Naive Bayes, or smoothing on HMMs (cf. Section 4.2). Boosting techniques can be viewed as a meta classification algorithm as they combine many weak classifiers to form one stronger classifier. For example, the popular Adaboost [FS95] computes a weighted sum of the weak classifiers’ outputs and is thus straightforward to realize based on HYPERPLANE, as also noted by Bost et al. [BPTG15].

Another interesting class of classification algorithms are nearest-neighbor methods [RB13]. To implement privacy-preserving variants in SHIELD, we can
reuse MAXARGMAX and need to extend our framework only by suitable distance computation protocols, e.g., the protocols proposed in [BCF+14] can be efficiently implemented using additive sharings and OT in our framework.

We expect more work will be necessary to realize other classification algorithms in our framework. For example, decision trees and random forests are popular classes of classification algorithms that are not directly supported in SHIELD but, being rooted in Boolean logic, could be efficiently implemented in GCs. Finally, the evaluation of ANN on large networks showed the limits of the numerical accuracy of our approach. Extending SHIELD to deep-learning, i.e., ANNs with orders of magnitude more hidden layers and neurons, requires improving the numerical accuracy of our building blocks.

Privacy-preserving learning: In our problem scenario, we assume that the classification model is already available. An interesting orthogonal problem is how to train these models on distributed data in a privacy-preserving manner (cf. Section 4.3.2). In the context of our work, two learning algorithms are especially interesting, i.e., gradient-descent used for training linear classifiers as well as neural networks and the Baum-Welch algorithm for maximum likelihood parameter estimation of HMMs. Secure protocols for both algorithms can reuse our number representation and many of our proposed primitives, e.g., Forward and Backward computation in the Baum-Welch algorithm can be realized directly using SHIELD while the Update phase requires an additional secure division primitive that is not yet realized in our framework but could potentially be designed along the ideas of our RESCALE protocol.

Use cases: We identify interesting use cases beyond ours: In privacy-preserving face recognition [EFG+09,OPJM10,SSW09] one party aims at privately matching a face against a database held by another party – the popular Viola Jones [VJ01] face detection algorithm can be directly implemented in SHIELD based on HYPERPLANE. A second interesting application area of SHIELD is privacy-preserving speech processing [PRRS13,PR13,SS07]. HMMs are ubiquitously used in speech processing [Rab89] and our secure FORWARD or VITERBI could directly be used to implement privacy-preserving keyword recognition or serve as major building block in a full-fledged automatic speech recognition pipeline.

Mobile deployments: We optimized our secure building blocks and classifiers for low latency and high throughput on standard desktop machines that are connected over a stable and reasonably fast network. However, many application scenarios of our designs involve users with mobile devices that communicate to services hosted in the cloud. Through the tremendously increased processing capabilities of modern smartphones, STC gradually becomes feasible on such devices processing-wise, but energy and communication overheads may quickly become bottlenecks, requiring whole different cost metrics for the design of secure protocols. As we noted in our problem analysis (Section 4.3), a general approach for catering to constrained environments is outsourcing and we thus propose efficient outsourcing schemes for the presented building blocks and classifiers in the next section. We furthermore motivate bandwidth-optimized secure protocols as an alternative strategy and optimize the communication overheads of selected building blocks.
4.5 Secure Classification and Pattern Recognition in Constrained Environments

The evaluation of our secure building blocks and classifiers (Section 4.4.8) and their application in the different use cases (Section 4.4.9) show that our approaches improve significantly upon the state of the art. The involved overheads are, as expected in the context of STC, orders of magnitude higher than equivalent algorithms operating on a single host on plaintext models and inputs. We especially observe significant communication overheads and a perceivably increase of the overall runtime due to network latency. While these overheads are manageable for desktop machines on fast and stable networks, they are likely to overtax mobile devices with limited processing power that communicate, e.g., over cellular networks with constrained bandwidth and higher latency.

Focusing on our third research questions (customizability), we present two strategies for tailoring our secure classifiers to mobile devices with limited processing and communication capabilities that possibly communicate over networks with limited bandwidth and high latency. We first present Bandwidth-Optimized protocols for Max and Argmax computation (BOMA) [ZHH+15], our approach to optimize the ubiquitously building block in our secure classification framework for bandwidth (Section 4.5.1). Even though BOMA’s significant reductions lead to overall shorter protocol runtimes in constrained networks, we notice that the remaining overheads may still overtax mobile users. As a second and completely orthogonal strategy (applied to FORWARD in [ZMR+17]), we thus propose to outsource computations to more capable peers, e.g., an untrusted computation cloud (Section 4.5.2).

4.5.1 Towards Bandwidth-optimized Secure Computations

We have observed significant communication overheads during the evaluation of our secure classifiers (Section 4.4.8) and of the real-world use cases (Section 4.4.9) that are likely to present a bottleneck to the overall protocol runtime in bandwidth-constrained networks. This is confirmed by our performance evaluation of STC frameworks [ZMHW15], where we find that communication overheads constitute the main performance bottleneck in constrained networks and has also been noted by others in [CMSA12, CADT14]. Another disadvantage of high bandwidth consumption is that it quickly depletes users’ data plans, inducing capped bandwidths, or significant costs for subsequent communication when communicating over cellular networks. Reducing communication overheads is even interesting beyond constrained environments, e.g., in large data centers where it may be cheaper to add computing power than to increase the capacity of network links. In this section, we thus consider the design of secure protocols that minimize bandwidth consumption.

We focus our efforts on the problem of computing the max and argmax, which are important building blocks not only in all of our classifier designs but also in many other STC applications ranging from nearest neighbor search [RB13] over auctions [KSS09] to biometric matching [EHKM11]. We first exactly quantify the communication complexity for state-of-the-art constructions and implementations of the (arg)max problem in the three major approaches to STC, i.e., GCs, GMW, and HE.
4.5. Secure Classification and Pattern Recognition in Constrained Environments

We then propose BOMA [ZHH15], a novel (arg)max protocol based on HE and OT that reduces communication overheads by orders of magnitude compared to prior works. In constrained networks with bandwidths ranging from 1 Mbit/s (e.g., Bluetooth, 3G) to 12 Mbit/s - 50 Mbit/s (e.g., WiFi, LTE), this translates to lower costs for end-users and even to faster runtimes than bandwidth-heavy algorithms.

4.5.1.1 Analysis of Efficient Secure Arg max Protocols

We analyze how the problem of securely computing max and arg max is solved in related work and in our improved protocol. Protocols marked with an asterisk solve only a restricted version of the arg max problem. \( n \) denotes the total number of values and \( t \) their bitlengths. \( s \) denotes the symmetric security level in bits, and \( T \) the equivalent asymmetric security level.

<table>
<thead>
<tr>
<th>Protocol</th>
<th>STC</th>
<th>Communication overhead</th>
<th>Rounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>* Kolesnikov’09 GC</td>
<td>( 2t(n-1)2t + (n+1)2t )</td>
<td>( \mathcal{O}(1) )</td>
<td></td>
</tr>
<tr>
<td>Huang’11 GC</td>
<td>( 2t(n-1)2t + nl + (n-1)2t )</td>
<td>( \mathcal{O}(1) )</td>
<td></td>
</tr>
<tr>
<td>Demmler’15 GC</td>
<td>( 3t(n-1)2t )</td>
<td>( \mathcal{O}(1) )</td>
<td></td>
</tr>
<tr>
<td>* Schoeller’15 GMW</td>
<td>( (n-1)4t - \lfloor \log_2(l) \rfloor - 2 + \lfloor \log_2(n) \rfloor)(2t+2) )</td>
<td>( \mathcal{O}(\log_2(n)\log_2(l)) )</td>
<td></td>
</tr>
<tr>
<td>Demmler’15 GMW</td>
<td>( (n-1)(5t - \lfloor \log_2(l) \rfloor - 2)(2t+2) )</td>
<td>( \mathcal{O}(\log_2(n)\log_2(l)) )</td>
<td></td>
</tr>
<tr>
<td>Erkin’09 HE</td>
<td>( (n-1)(2t+8 + 10/C)T )</td>
<td>( \mathcal{O}(\log_2(n)) )</td>
<td></td>
</tr>
<tr>
<td>BOMA’15 HE + OT</td>
<td>( (n-1)(4+6/C)T + 2nT/C + n(l+\sigma) + 3\log_2(n)t )</td>
<td>( \mathcal{O}(\log_2(n)) )</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.1 Communication complexity in bits of secure protocols for the arg max problem in related work and in our improved protocol. Protocols marked with an asterisk solve only a restricted version of the arg max problem.

We proceed to analyze the most efficient solutions based on GCs, GMW, and HE (we are not aware of any approach that uses additive arithmetic sharings).

**Problem definition.** Given a set of \( n \) arguments \( X = (x_1, \ldots, x_n) \) and \( n \) corresponding function values \( Y = (y_1 = f(x_1), \ldots, y_n = f(x_n)) \), the task is to find \( x^* \) and/or \( f(x^*) \) such that \( f(x^*) \leq f(x) \) \( \forall i \leq n \). We refer to \( y^* = f(x^*) \) as the max and to \( x^* \) as arg max. We assume that the inputs \( X \) and \( Y \) are already available to the two parties, the user \( U \) and service provider \( S \), as garbled values (GC), secret Boolean sharings (GMW), or encryptions (HE) and the output should be protected accordingly. This represents the usual case where the (arg)max algorithm is used as a building block within another secure computation, e.g., as in our classifier designs or nearest neighbor search [EHKM11, RB13]. Our results, however, extend to other settings, e.g., where \( U \) and \( S \) each hold half of the inputs and the (arg)max should be obtained in clear only by one party.

**Overview.** In the following, we analyze the most efficient (arg)max protocols based on GC, GMW, and HE. Their respective communication and round complexity is summarized in Table 4.1. The last row in Table 4.1 shows the complexity for our improved protocol which we present and analyze in Section 4.5.1.2. Approaches marked with an asterisk restrict the value space of the argument of the maximum to \( \{0, \ldots, n-1\} \). This restriction allows for efficiency improvements but does not fully meet our problem definition as it would require further computation to realize the full range of applications. E.g., in biometric access control applications, the arg max is not only the index of a user but her full profile including access rights [EHKM11].
Nevertheless, we include them in our analysis since they indicate useful lower bounds. Note that we also use the general (arg)max version in the implementation of our classifier designs to allow returning arbitrary classes and not just their indices.

**Secure Arg max based on GCs.** The communication complexity of GC-based (arg)max protocols is dominated by i) the overhead for the input transfers (via OT), ii) the size of the underlying circuit, and iii) the chosen garbling scheme. The overheads for i) do not occur when the arg max protocol is used as a building block within another secure computation as in our problem definition. To quantify the overheads for ii), we analyze the most efficient circuit constructions and their respective sizes below. Regarding iii), we assume the most recent ‘Half Gates’ garbling scheme [ZRE15] which requires only $2t$ bits of communication per gate and is optimal with regards to communication overhead. The combination of the most efficient circuit construction with an optimal garbling scheme yields lower bounds for the communication complexity for GC-based (arg)max protocols.

Kolesnikov et al. [KSS09] present a widely used (arg)max circuit construction that determines the max in a pairwise tournament tree fashion and constructs the arg max while traversing down the tree. The circuit has a size of $2l(n-1)$ gates for finding the max and $n+1$ gates for constructing the arg max, resulting in a total communication overhead of $(2l(n-1) + (n+1))2t$ bit. In this construction, the arg max is limited to the values $\{0, ..., n-1\}$.

Huang et al. [EHKM11] propose a different construction to overcome the limitation of the arg max value space, based on the observation that encoding complex data structures into the GC is very expensive. Hence, while using the same circuit of size $2l(n-1)$ for finding the max as [KS08a], they replace the arg max functionality with a custom backtracking protocol. This protocol exchanges a fully encrypted backtracking tree from which exactly one path can be decrypted using the wire keys obtained during the evaluating of the garbled max circuit. Note that $U$ is allowed to recover the arg max in clear in [EHKM11], but the arg max must be garbled to use this construction as a secure building block. This results in an overhead of $nl$ for encrypting $n$ $l$-bit arg max values in the tree’s leaves and $(n-1)2l$ bit for the two wire keys in each of the $n-1$ inner nodes of the backtracking tree.

Finally, Demmler et al. [DSZ15] present the ABY framework for STC, which implements the max circuit with $2l(n-1)$ gates proposed by Kolesnikov et al. [KSS09] but does not supply the arg max\(^{15}\). We extend their implementation by adding $1$ MUX gates per comparison which propagate the $l$ bit arg max to obtain a second construction that fulfills our general problem definition. A single $l$ bit MUX gate can be realized using $l$ Non-XOR gate, hence transmitting the complete circuit requires $3(n-1)2t$ bit of communication.

**Secure Arg max based on GMW.** Schneider et al. [SZ13] present depth-optimized circuit constructions for GMW. Their (arg)max circuit is based on the construction by Kolesnikov et al. [KSS09] but replaces the size-optimized $l$-bit comparators with depth-optimized comparators that consists of about two times more gates but has only logarithmic instead of linear depth in $l$. The circuit has $(n-1)(4l - \lceil \log_2(l) \rceil - 2) + \ldots$

\(^{15}\)This refers to the ABY version as of June 2015, when we developed BOMA. At the time of writing this thesis, the ABY framework has been extended with a secure argmax protocol similar to the construction that we added in 2015 for the measurements presented in [ZHH+15].
4.5. Secure Classification and Pattern Recognition in Constrained Environments

[$\log_2(n)$] gates and a depth of $O(\log_2(n) \log_2(l))$ [SZ13]. To evaluate a single gate, $U$ and $S$ exchange 2 bit and engage in one random OT (a specialized form of general OT where the secrets held by the sender are random values [ALSZ13]) at the costs of 2$\ell$ bit of communication. Since this construction bases on Kolesnikov’s [KSS09], it has the same limitations regarding the possible arg max values.

The $ABY$ framework presented by Demmler et al. [DSZ15] also implements GMW together with the depth-optimized maximum circuit presented in [SZ13] but without the arg max logic. Again, we extend the implementation using $\text{MAX}$ gates to relay the arg max along with the computation of the max. In contrast to Schneider’s proposed circuit, this construction fulfills our problem definition. The circuit has $(n - 1)(5 - \log_2(l)) + 2$ gates and a depth of $O(\log_2(n) \log_2(l))$.

**Secure Arg max based on HE.** Most HE-based (arg)max constructions use the Damgard-Geisler-Kroigaard (DGK) cryptosystem and comparison protocol proposed in [DGK08], that compares two Paillier-encrypted integers $[y_1]$ and $[y_2]$ and returns the result as an encrypted bit $[y_0 \leq y_2]$. It has been used in a variety of applications, e.g., face recognition [EFG+09], recommender systems [EVTL12], bioinformatics [FDH+11], and clustering [EVTL09].

The most efficient (arg)max protocol that fulfills our problem definition has been proposed by Erkin et al. [EFG+09]. The authors arrange comparisons of values in a pairwise tournament fashion and propagate the max and arg max by multiplying with the encrypted comparison bit, i.e., $[\text{max}(y_1, y_2)] = [y_1 \geq y_2] \cdot [y_1 - y_2] \oplus [y_2]$ and $[\text{arg max}(y_1, y_2)] = [y_1 \geq y_2] \odot [x_1 - x_2] \oplus [x_2]$. Each comparison needs to exchange 2$\ell$ DGK ciphertexts and three Paillier ciphertexts. To propagate the max and arg max through one comparison, two ciphertext multiplications are required, causing transmission overheads of six Paillier ciphertexts [KSS10]. This amounts to a communication complexity of $(n - 1)(2\ell + 10)/T$ bit. While Erkin et al. [EFG+09] do not consider ciphertext packing in their original work, we are able to improve the communication complexity of their approach to $(n - 1)(2\ell + 8 + 10/C)/T$ bit by packing the Paillier ciphertexts sent from the server provider to the user (the achieved compression factor is $C = [T/(l + \kappa + 2)]$).

4.5.1.2 Bandwidth-optimized Max and Arg max

To address the significant problems arising from high communication overheads of STC protocols in constrained environments, we propose BOMA, a bandwidth-optimized protocol for the secure (arg)max computation on homomorphically encrypted inputs. We emphasize that by using efficient conversions between encrypted, shared, and garbled values [DSZ15], BOMA can substitute the $\text{MAX} \leftarrow \text{ARGMAX}$ primitive used in all our classifier designs or serve as a building block to improve other STC frameworks, e.g., [DSZ15, HEKM11]. Following, we first present an efficient comparison protocol and then build an efficient (arg)max protocol on top. Of course, these protocols are straightforward to adopt to the symmetric (arg)min problem.

**Efficient secure comparisons.** As an important building block, we use Kersschbaum’s multi-party comparison protocol [KB-H09]. The core idea is to compute the encrypted distance $[d] = [a - b]$ between two homomorphically encrypted inputs $[a]$ and $[b]$ and let all participants blind the distance $[d]$ while preserving its sign before decrypting it and deciding the comparison.
Protocol 4.13 Bandwidth-optimized \((\arg)\text{max}\) protocol: The max and its permuted index are calculated. Then, \(U\) and \(S\) run an OT protocol to unblind the \(\text{arg}\max\).

We adapt the multi-party protocol from [KBr09] to our two-party setting: \(S\) selects two large random numbers \(r_1\) and \(r_2\) in \([0,1)^4\) with \(r_1 > r_2\) and computes \([\overline{d}] = [r_1 \cdot (a - b) - r_2]\) under encryption. Note that the multiplicative blinding preserves the sign, i.e., \(d \geq 0 \Leftrightarrow a \geq b\). \(S\) then sends the blinded encrypted distance \([\overline{d}]\) to \(U\) who decrypts it and sends back the encrypted comparison bit \([\overline{a}] \geq \overline{b}\). To prevent \(U\) from learning the real outcome of the comparison, \(S\) chooses at random to compare \(a \geq b\) or \(b \geq a\) and flips the received comparison bit \([\overline{a} \geq \overline{b}]\) accordingly by computing \([1 + (a - \overline{b})] \) under encryption. Our two-party version of Kerschbaum’s comparison exchanges only 2 ciphertexts, i.e., \(4T\) bit, between \(U\) and \(S\).

**Efficient secure \((\arg)\text{max}\).** As a first improvement, we replace the DGK comparison protocol in Erkin’s \((\arg)\text{max}\) algorithm [EFG+09] with our two-party version of Kerschbaum’s comparison protocol. By simply using a more efficient comparison protocol, we reduce the communication complexity to \((M - 1)(6 + 10/C)T\) bit, i.e., save \((M - 1)(2T + 2C T)\) bit (cf. Table 4.1). This represents a significant reduction by 90% to 97% depending on the chosen security level \(t\) and bitlength \(t\).

We now further processing and communication overheads of this construction (Protocol 4.13). In the max finding phase, we determine the max \([y^*] = [f(x^*)]\) in pairwise comparisons arranged in a tournament fashion as before in [EFG+09]. However, we significantly reduce processing overheads and save \(2 \log_2(n)\) rounds of communication by interleaving comparison and selection steps, thereby shaving off costly ciphertext multiplications. Our second significant improvement is due to the observation that \(U\) learns the index of the max in the input vector as a byproduct of this protocol. With this information, we construct an efficient OT-based protocol for the second phase, the \(\text{arg max finding phase}\), in which \(U\) helps \(S\) to obtain an encryption of the \(\text{arg max}\) \([\overline{z}^*]\). In the following, we describe the two phases in detail.
Finding the maximum $y^*$: At the beginning, $S$ holds the encrypted values $[y_1], \ldots, [y_n]$ and corresponding arguments $[x_1], \ldots, [x_n]$ and applies the same permutation $\pi$ to both. This permutation prevents information leakage to $U$ but has no effect on the outcome of the computation. For reasons of simplicity, we thus leave $\pi$ out in the following. At the core of the maximum finding phase are the batched pairwise comparisons according to our two-party version of Kerschbaum’s comparison protocol. $S$ computes the distances $[d_i]$ over the $n/2$ pairs of values $[y_i]$ and $[y_{i+1}]$ for $i = 0, 2, \ldots, n-2$ and blinds the distances as well as the values. $S$ then packs and sends the blinded distances together with the blinded values $[\bar{y}_i]$ and $[\bar{y}_{i+1}]$ to $U$. $U$ decrypts each distance $d_i$, determines the sign, and encrypts the binary result of the comparison $b_i = (d_i \geq 0)$. Based on the outcome of the comparison, $U$ chooses $[\bar{y}_i]$ or $[\bar{y}_{i+1}]$ and re-randomizes it by adding $[0]$, i.e., a fresh encryption of zero. Finally, $U$ sends the encrypted comparison result $[b_i]$ together with the re-randomized winner of each comparison $[\bar{y}_{i+1}]$ back to $S$. Note that $S$ cannot distinguish which elements were received due to the re-randomization and semantic security of the cryptosystem. After unblinding the received values, $S$ now holds encryptions of the $n/2$ smaller values of the previous comparisons and repeats the previous steps $\log_2(n) - 1$ times until only the max $[y^*]$ remains. Our way of interleaving comparison and selection steps not only makes ciphertext packing more efficient, but significantly reduces processing costs and saves two communication rounds per level of the comparison tree, i.e., $2\log_2(n)$ rounds in total, as compared to the construction of Erkin et al. [EFG+09] which implements selection steps via costly ciphertext multiplication.

Finding the maximum argument $x^*$: We further significantly improve the communication overhead by the observation that during the maximum phase it is easy for $U$ to keep track of the outcomes of the comparisons and thereby obtain the index $j^*$ of the max in the permuted vector $Y$. This knowledge is not a security violation since the permuted index does not disclose any information to $U$ who does not know the permutation $\pi$. In the argmax finding phase, $U$ now helps $S$ to obtain an encryption of the argmax $[x_{j^*}]$. $S$ first blinds all encrypted arguments $[x_i]$ for $j = 1, \ldots, n$ individually by subtracting random values $r_j \in \{0,1\}^{l+n}$ and a second time with the same random value $r \in \{0,1\}^{l+n}$. $S$ sends the double blinded arguments to $U$ using ciphertext packing to reduce communication overheads. $U$ and $S$ subsequently engage in $1-n$ OTs $\text{OT}_{1-n}$, after which $U$ obtains $r_{j^*}$, the individual blind of the argmax, without $S$ learning $j^*$. Then, $U$ removes this blind from $[\bar{x}_{j^*}]$ by adding $r_{j^*}$ (which automatically re-randomizes the value) and sends the value (still blinded by $r$) to $S$. Finally, $S$ removes the second blind by adding $r$ and obtains an encryption of the argmax $[\bar{x}_{j^*}]$ (remember that an encryption of the max $[y^*]$ has already been obtained in the maximum finding phase).

4.5.1.3 Security Discussion

We now show that our proposed protocol is secure in the semi-honest adversary model. In particular, we show that i) $U$ learns nothing and ii) $S$ only learns an encryption of the max and argmax but nothing else.

Security against $U$. $U$ learns nothing about the values $y_i$ from the received $\bar{y}_i$ since they are additively blinded by random numbers $r_i \in \{0,1\}^{l+n}$. Furthermore, $U$ learns
nothing from $d_i$ about the distance $d_i$ between $y_i$ and $y_{i+1}$ due to the multiplicative and additive blinding. From the messages received in the argmax finding phase, $U$ learns nothing since the arguments $x_j$ are additively blinded using $r_j, r \in \{0,1\}^{l+\kappa}$. The security of the unblinding steps follows from the security of OT. After the OT has finished, $U$ learns $x_j - r$ which however is still additively blinded. Finally, $U$ learns the index of the maximum. Due to the random permutation $\pi$ applied by $S$ at the start, this knowledge is useless to $U$ as long as $\pi$ is kept secret.

Note that we use statistical blinding, i.e., with low probability $\sim 1/2^\kappa$ $U$ learns a small amount of information about the magnitude of the blinded values. We can achieve perfect security against $U$ by choosing $\kappa = T$ and substituting Kerschbaum’s statistically secure protocol [KBdH09] with a perfectly secure protocol, e.g., [Veu12]. However, this increases the communication overhead by orders of magnitude.

**Security against $S$.** From the messages received in the max finding phase, $S$ learns the encrypted outcome of the comparison (the comparison bit and larger element). Due to the semantic-security of the Paillier cryptosystem and the re-randomization, $S$ can neither decide whether $[b_i]$ is an encryption of 0 or 1 nor distinguish $[\bar{y}_i]$ from $[\bar{y}_{i+1}]$. Again, $S$ learns nothing from the OT in the argmax phase due to the security of the employed OT primitives [NP05,NP01,IKNP03,ALSZ13]. Finally, $S$ receives $[\overline{x}_j]$ which it cannot distinguish from the other arguments $[\overline{x}_{j'}]$ due to the semantic security of the cryptosystem. Since $S$ can always try to break encryption to learn all inputs, we can only achieve computational security against $S$.

### 4.5.1.4 Evaluation

We analyze the communication complexity of our optimized (arg)max protocol then compare its communication overheads against those of prior works. Since our protocol trades increased local processing for a significant reduction in communication overhead, we show afterwards that processing overheads are still reasonable. (Although runtime was not our primary optimization goal, the reduction of communication overheads outweighs the increased local processing in networks with limited bandwidth and renders our protocol even faster than prior works).

**Communication Complexity**

The max finding phase of BOMA costs $(n-1)(4+6/C)T$ bit, where $C = \lfloor T/(l+\kappa+2) \rfloor$ is the ciphertext packing rate. The arg max finding phase costs $n 2T/C$ bit for transferring the blinded arguments and $n(l+\kappa) + 3 \log_2(n)t$ bit for the OT of the blind $r_j$ (excluding the costs for the base OTs which are one time overheads of only $t^2 + tT$ bit of communication [NP05,NP01] that amortize over multiple runs).

**Comparison of Communication Overheads**

We compare the communication overhead of BOMA to the analyzed (arg)max protocols of related works. For Kolesnikov’09 [KSS09], Evans’11 [EHKM11], and Schneider’13 [SZ13], no implementation is available so that we need to estimate their communication overheads based on the theoretical complexities (cf. Table 4.1).
Kolesnikov’09 and Schneider’13 realize the constrained argmax functionality and hence only represent lower bounds. For ABY-YAO’15 and ABY-GMW’15, we measure the listed results using the C++ ABY framework [DSZ15, aby15] which we extend with the missing argmax circuits. We implement BOMA (our own protocol), in Python 2.7 and use msgpack [Mes] for serializing messages. Additionally, we re-implement the (arg)max algorithm by Erkin et al. [EFG+09], since the authors’ implementation [Tod13] provides neither network support nor ciphertext packing.

For all available implementations, we initially compare the measured (using tcpdump) and theoretical communication overhead to analyze i) the accuracy of our theoretical complexity estimates and ii) the realization of these complexities in the actual implementations. We find that the measured overhead exceeds our estimate by at most 0.5% for ABY-YAO, 1.5% for ABY-GMW, 2% for BOMA (our own protocol), and by less than 6% for our re-implementation of Erkin’s protocol. The significant deviation for Erkin’s protocol stems from the fact that our theoretical complexity estimates do not consider a decrease in packing efficiency when only few ciphertexts are left at the last levels of the comparison tree.

Figure 4.18 shows the communication overheads for different bitlengths and security levels. The detailed measurements and the factors of improvement are given in Table 4.2. BOMA achieves a significant reduction in communication overheads in all settings and even when compared to the restricted argmax circuits by Kolesnikov’09 (GC) and Schneider’13 (GMW). Specifically, BOMA achieves the largest reductions for small security levels and high bitlengths of the input. For higher security levels, the relative improvement in comparison to the GC-based and GMW-based approaches decreases while still outperforming said approaches. Conversely, larger input bitlengths benefit BOMA since packing efficiency for communication from $S$ to $U$ only degrades slightly but communication from $U$ to $S$, which cannot be packed, remains the same since a single $l$ bit value always fits into one ciphertext for $l \ll T$.

In contrast, the communication overheads in GC and GMW scale linearly in $l$. Finally, BOMA outperforms Erkin’09 by at least one order of magnitude in every setting.
### Security level \( l \)

<table>
<thead>
<tr>
<th>Bitlength ( n )</th>
<th>short</th>
<th>medium</th>
<th>long</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>64</td>
<td>128</td>
<td>64</td>
</tr>
<tr>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
</tr>
<tr>
<td>256</td>
<td>256</td>
<td>256</td>
<td>256</td>
</tr>
</tbody>
</table>

* Koles’09 
(GC) 1.30 × 2.58 × 5.13 × 1.82 × 3.61 × 7.19 × 2.08 × 4.12 × 8.22

<table>
<thead>
<tr>
<th>Evans’11</th>
</tr>
</thead>
<tbody>
<tr>
<td>(GC) 1.93 × 3.70 × 6.29 × 1.54 × 2.99 × 5.54 × 1.23 × 2.40 × 4.56</td>
</tr>
<tr>
<td>(GC) 2.40 × 4.62 × 7.86 × 1.92 × 3.73 × 6.92 × 1.53 × 2.99 × 5.69</td>
</tr>
</tbody>
</table>

* Schneider’13
(GMW) 3.94 × 7.49 × 12.71 × 3.14 × 6.03 × 11.15 × 2.50 × 4.83 × 9.17

<table>
<thead>
<tr>
<th>ABY-GMW’15</th>
</tr>
</thead>
<tbody>
<tr>
<td>(GMW) 4.78 × 9.41 × 16.22 × 3.79 × 7.54 × 14.16 × 3.02 × 6.03 × 11.63</td>
</tr>
</tbody>
</table>

* Erkin’09
(HS) 9.83 × 18.46 × 35.78 × 19.02 × 35.84 × 69.53 × 28.22 × 53.23 × 103.27

<table>
<thead>
<tr>
<th>BOMA’15</th>
</tr>
</thead>
<tbody>
<tr>
<td>14.60 × 26.50 × 43.83 × 16.14 × 29.67 × 53.55 × 16.72 × 30.94 × 57.30</td>
</tr>
</tbody>
</table>

Table 4.2: Communication overhead in MB and factors of improvement for computing the \( \text{argmax} \) over \( n = 1000 \) elements for varying security levels and bitlengths of the inputs. Gray rows denote lower bounds for the respective approaches derived from the theoretical communication complexity according to Table 4.1 – all other values are measured. Approaches marked with an asterisk realize only a restricted \( \text{argmax} \) functionality.

### Comparison of Runtime Overheads

The previous results show that we have reached our main goal of reducing communication overheads. However, we achieved this goal by trading communication against processing overheads. Thus, we now measure the runtime for ABY-YAO, ABY-GMW, Erkin’09, and compare it against the runtime of BOMA to quantify this trade-off. This allows us to prove two points: First, that the processing overheads for BOMA are still reasonable, and, second, that the significant reduction of communication even leads to overall shorter runtimes in networks with limited bandwidth.

We measure the respective approaches in a local setup between a desktop client (Intel i7, 8 cores at 2.93 GHz and 4 GB RAM) and a server (Intel Xeon, 16 cores at 2.6 GHz and 32 GB RAM) connected on a 1 Gbit/s LAN of which we vary the bandwidth between 1 Mbit/s and 10 Mbit/s using netem (we briefly discuss the impact of network latency afterwards). We choose a desktop client instead of a mobile client to maintain comparability as no ABY implementation for Android or iOS exists. While ABY is fully threaded and thus allocates all cores on the client device, we deliberately parallelize only our server-side functionality implementation to emulate processing resources comparable to those of a mobile device, e.g., a smartphone. The reported numbers are thus a rather pessimistic comparison. Since all overheads scale linearly in \( n \), we fix \( n = 1000 \) which is sufficiently large to eliminate small scale effects but also maintains short runtimes to allow a repeated, thorough evaluation, i.e., averaging all results over 30 runs.
4.5 Secure Classification and Pattern Recognition in Constrained Environments

Figure 4.19 Runtime overhead in seconds of different secure argmax protocols, ABY-YAO’15 (AY), ABY-GMW’15 (AG), and ours, BOMA’15 (BO), for computing the argmax over $n = 1000$ elements for varying security levels $t \in \{\text{short, medium, long}\}$, bitlengths $l \in \{32, 64, 128\}$ (left), $l = 64\text{bit (middle)}$, and $l = 128\text{bit (right)}$, and network bandwidths of 1 Mbit/s to 10 Mbit/s.

Network Bandwidth. We first vary the bandwidth using netem on the middlebox between 1 Mbit/s and 10 Mbit/s, which are representative choices according to the global mobile connection speeds presented in Akamai’s report on the state of the Internet [TSM+17]. Figure 4.19 compares the runtimes of existing approaches and our (arg)max protocol (BOMA) for bitlengths $l \in \{32, 64, 128\}$ and short, medium, and long-term security levels. The complete measurements and factors of improvement are provided in Table 4.3.

For short-term security (green bars in Figure 4.19), BOMA performs in the same order of magnitude as ABY-YAO and ABY-GMW at a bandwidth of 10 Mbit/s, being slightly slower for smaller inputs ($l = 32\text{bit}$) and slightly faster for longer input lengths ($l = 128\text{bit}$). Reducing to 1 Mbit/s, the runtime of BOMA only doubles while the runtimes of ABY-YAO and ABY-GMW increase by roughly one order of magnitude. In this scenario, the communication overhead clearly dominates the runtime of these approaches and BOMA hence outperforms them. As indicated by the theoretical complexity, the approach by Erkin et al. [EFG+09] is by orders of magnitude slower. Even in a most favorable setting, i.e., $l = 32\text{bit}$, short-term security, and no bandwidth-constraints, the protocol already ran for almost 1 min—we waived further measurements due to the prohibitive runtime of this approach.

Increasing to medium security (blue bars in Figure 4.19) impacts BOMA more than ABY-YAO or ABY-GMW which is due to the use of asymmetric versus symmetric cryptography. Here, BOMA roughly matches the runtime of ABY-YAO for all bitlengths $l$ at bandwidths between 1 Mbit/s and 5 Mbit/s and ABY-GMW between 2 Mbit/s and 10 Mbit/s. Still, we observe that BOMA outperforms these approaches in slow networks of 1 Mbit/s to 2 Mbit/s and for higher bitlengths $l \in \{64, 128\}$.

Increasing to long-term security (red bars in Figure 4.19), we observe that processing overheads begin to dominate the performance of BOMA, i.e., the relative difference of the performance at bandwidths of 1 Mbit/s and 10 Mbit/s is much smaller than for shorter security levels. Contrarily, for ABY-GMW and ABY-YAO, which exhibit very low processing overheads, bandwidth restrictions continue to dominate the overall protocol runtime. In comparison, BOMA still outperforms ABY-YAO up to bandwidths of 2 Mbit/s and ABY-GMW up to 5 Mbit/s for $l = 128\text{bit}$ in-
reduces communication overheads by 18\% to 98\% compared to prior works. The

Table 4.3. Measured overall runtimes in seconds of state-of-the-art implementations of GC-, GMW- and HE-based argmax protocols for computing the argmax over \( n = 1000 \) elements for varying security levels, bitlengths of the inputs, and network bandwidths.

<table>
<thead>
<tr>
<th>Security level ( l )</th>
<th>short</th>
<th>medium</th>
<th>long</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bitlength ( l )</td>
<td>32</td>
<td>64</td>
<td>128</td>
</tr>
<tr>
<td>1 Mbit/s</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.9×</td>
<td>16.70</td>
<td>33.50</td>
<td>67.02</td>
</tr>
<tr>
<td>5 Mbit/s</td>
<td>3.38</td>
<td>6.92</td>
<td>13.86</td>
</tr>
<tr>
<td>10 Mbit/s</td>
<td>1.76</td>
<td>3.72</td>
<td>7.19</td>
</tr>
<tr>
<td>0.4×</td>
<td>0.7×</td>
<td>1.2×</td>
<td>0.2×</td>
</tr>
<tr>
<td>0.6×</td>
<td>1.2×</td>
<td>2.1×</td>
<td>0.4×</td>
</tr>
<tr>
<td>1.0×</td>
<td>0.6×</td>
<td>1.2×</td>
<td>0.2×</td>
</tr>
<tr>
<td>GC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 Mbit/s</td>
<td>27.61</td>
<td>56.15</td>
<td>113.45</td>
</tr>
<tr>
<td>3.1×</td>
<td>38.28</td>
<td>77.95</td>
<td>157.46</td>
</tr>
<tr>
<td>5 Mbit/s</td>
<td>5.74</td>
<td>11.56</td>
<td>23.10</td>
</tr>
<tr>
<td>1.1×</td>
<td>7.86</td>
<td>15.84</td>
<td>31.79</td>
</tr>
<tr>
<td>10 Mbit/s</td>
<td>3.12</td>
<td>6.09</td>
<td>11.90</td>
</tr>
<tr>
<td>0.6×</td>
<td>4.17</td>
<td>8.25</td>
<td>16.33</td>
</tr>
<tr>
<td>0.8×</td>
<td>1.1×</td>
<td>1.9×</td>
<td>0.4×</td>
</tr>
<tr>
<td>1.0×</td>
<td>0.9×</td>
<td>0.5×</td>
<td>0.2×</td>
</tr>
<tr>
<td>0.2×</td>
<td>0.3×</td>
<td>0.5×</td>
<td>0.2×</td>
</tr>
<tr>
<td>ABY-GMW</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 Mbit/s</td>
<td>8.96</td>
<td>9.41</td>
<td>11.00</td>
</tr>
<tr>
<td>3.1×</td>
<td>22.74</td>
<td>21.88</td>
<td>27.20</td>
</tr>
<tr>
<td>5 Mbit/s</td>
<td>5.33</td>
<td>5.72</td>
<td>6.74</td>
</tr>
<tr>
<td>1.1×</td>
<td>15.34</td>
<td>16.54</td>
<td>19.39</td>
</tr>
<tr>
<td>10 Mbit/s</td>
<td>4.88</td>
<td>5.34</td>
<td>6.24</td>
</tr>
<tr>
<td>0.6×</td>
<td>14.43</td>
<td>15.69</td>
<td>18.41</td>
</tr>
<tr>
<td>0.8×</td>
<td>3.29</td>
<td>3.36</td>
<td>3.89</td>
</tr>
<tr>
<td>BOMA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 Mbit/s</td>
<td>42.86</td>
<td>45.59</td>
<td>51.57</td>
</tr>
<tr>
<td>3.1×</td>
<td>45.59</td>
<td>48.82</td>
<td>51.57</td>
</tr>
<tr>
<td>5 Mbit/s</td>
<td>32.46</td>
<td>34.92</td>
<td>40.23</td>
</tr>
<tr>
<td>10 Mbit/s</td>
<td>31.29</td>
<td>33.68</td>
<td>38.99</td>
</tr>
</tbody>
</table>

puts. For \( l \in \{32, 64\} \) and larger bandwidths, BOMA’s performance is slower but in general lies within the same order of magnitude as ABY-YAO and ABY-GMW.

Network Latency. Besides the bandwidth limitations considered so far, the network’s latency also has a non-negligible impact on the performance of secure \((arg)\max\) protocols – especially on those protocols that require multiple sequential rounds of communication. The analyzed GMW-based protocols have a high round complexity of \( O(\log_2(n) \log_2(l)) \) and are thus slowed down the most. Assuming a relatively high latency of 100ms, the overall runtime for computing the \((arg)\max\) over \( n = 1000 \) and \( l = 32 \) bit to \( l = 128 \) bit values is increased by 5s to 7s. Under the same assumptions, BOMA’s runtime increases only by approximately 1s due to its one order of magnitude lower round complexity of \( O(\log_2(n)) \). In contrast, GC-based protocols experience nearly no increase in runtime due to their constant round complexity. Hence, settings with higher network latencies favor BOMA over GMW-based approaches while the disadvantage compared to GC-based approaches remains small.

4.5.1.5 Summary and Future Work

With BOMA, we advocate elevating communication overheads to a primary optimization goal and considered the exemplary problem of securely computing the \((arg)\max\), which is a building block of all our previous classifier designs and used in many other STC applications. Our novel \((arg)\max\) protocol based on HE and OT reduces communication overheads by 18\% to 98\% compared to prior works. The
4.5. Secure Classification and Pattern Recognition in Constrained Environments

evaluation results support our initial observations that communication overheads can significantly decrease performance of STC protocols and must be optimized when used in mobile environments. Specifically, the improved performance of BOMA under the reduced network speeds of 3G or LTE networks, e.g., 1 Mbit/s to 10 Mbit/s as found in typical current urban scenarios [TSM +17], highlights the suitability of our protocol for such environments.

Considering that our classifiers also require several other building blocks beyond arg max, further work is necessary to optimize also their communication overheads. To this end, general approaches would be preferable over the special purpose protocol that we proposed in this section. General approaches to reduce communication overheads of secure protocols (in a trade-off for increased processing) based on look-up tables [DKS +17] or optimized garbling schemes [ZRE15] have been proposed only recently. In this context, we view our work on bandwidth-constrained secure computations as a first proposal for elevating communication overheads to a primary optimization goal – a point-of-view that slowly gains more attention as increasing capabilities of modern smartphones render STC feasible in mobile scenarios and require to account for users’ limited data rates and volumes.

Critically reviewing the overheads of BOMA and those of other related proposals that trade off communication against local processing [ZRE15, DKS +17], we argue that the involved overheads may still be prohibitive for mobile applications and more radical approaches are required. In the next section, we thus propose a second alternative and fully compatible strategy for tailoring STC to challenged environments.

4.5.2 Secure Outsourcing to Untrusted Computation Clouds

Based on the observation that even highly optimized STC protocols may be too costly to be executed directly on mobile devices, we investigate a completely orthogonal strategy in this section, i.e., the secure outsourcing of the execution of STC protocols to untrusted computation clouds. We formalize the considered scenario, then show how to outsource all of SHIELD’s building blocks and classifiers, and evaluate the overheads for outsourcing showing that this strategy is feasible even for very constrained devices and networks. As an additional advantage, the outsourcing approach affords disruption tolerance, i.e., U and S need to be online and available only at the start and end of the computation – clearly a desirable property for mobile scenarios where intermittent connectivity is common.

4.5.2.1 Outsourcing Scenario

Figure 4.20 illustrates the considered outsourcing model: We assume that both user U and service provider S need to outsource computations – the case where only one party outsources computations is straightforward from there. We allow both parties to individually choose separate cloud peers to outsource to, denoted $C_U$ and $C_S$, respectively. The two parties then execute the outsourcing which consists of four separate steps, i) preprocessing, ii) outsourcing, iii) result computation, and iv) postprocessing: In the preprocessing phase (Step 1), $U$ and $S$ may shortly interact directly to prepare the subsequent steps based on their respective inputs, e.g., the
feature vector \( \vec{x} \) or classification model \( M \). In the outsourcing phase (Step 2), \( U \) and \( S \) set up \( C_U \) and \( C_S \) with the required data. The main phase (Step 3) is the outsourced computation of the results, i.e., the secure classification or pattern recognition task, between \( C_U \) and \( C_S \) without any involvement of \( U \) or \( S \). Finally, the results are returned to \( U \) and \( S \) who may execute a postprocessing protocol (Step 4). Note that our scenario and outsourcing model are strictly different from the assumptions and model used in the context of \( \text{FHE} \), where a single user (often referred to as the data holder) securely outsources computations including all required inputs to a single computation cloud, e.g., as in Dowlin et al.’s Cryptonets [DGBL+16].

To present a real alternative to mobile users, an outsourcing protocol must fulfill the following intuitive requirements: First, the overheads for all protocol steps that involve \( U \) or \( S \) should be minimized, i.e., preprocessing, outsourcing, and postprocessing. Second, the overheads for the outsourced result computation must be feasible for the cloud peers. Third, outsourcing and outsourced result computations must not violate our security and privacy guarantees, i.e., the computation parties \( C_U \) and \( C_S \) must remain completely oblivious of the inputs and outcome of the computation as indicated by the trust spheres in Figure 4.20. For the definition of security, we stick to the security model of this chapter and assume that \( C_U \) and \( C_S \) are semi-honest and do not collude, which is a reasonable assumption for typical infrastructure-as-a-service cloud providers that have strong incentives to guard their reputation and execute outsourced computations correctly [KMRS14,PSSZ15,ATD15].

We now examine how our proposals for secure classification and pattern recognition can be outsourced in this model. We first present outsourcing schemes for our building blocks \texttt{Rescale}, \texttt{ScalarProduct}, \texttt{MaxArgmax}, \texttt{FunctionApprox}, \texttt{EvalProbMass}, \texttt{Gaussian}, and \texttt{LogSum} – outsourcing the secure classifiers is then straightforward as they mainly compose these building blocks.

### 4.5.2.2 Outsourcing our Building Blocks

\texttt{Rescale} (Protocol 4.1), \texttt{ScalarProduct} (Protocol 4.3), and \texttt{MaxArgmax} (Protocol 4.4) take only additive shares as inputs, learn nothing from intermediate values
(as discussed in Section 4.4.7), and output the result again as additive shares. This renders the secure outsourcing of these primitive very easy and efficient. Without any preprocessing, $U$ and $S$ just send their individual shares to $C_U$ and $C_S$ which execute the protocol exactly as proposed and just return the shares of the result back to $U$ and $S$ who may postprocess them as desired, e.g., exchange them to recombine the result. **Gaussian** (Protocol 4.9) can be outsourced in the same way: Instead of using dummy sharing on their respective inputs, $U$ and $S$ share their inputs directly to $C_U$ and $C_S$ who compute the protocol as-is.

**FunctionApprox** and its specialization, **LogSum**, are more difficult to outsource since their initial designs requires one party to know the function in clear that is being approximated. When the knowledge of that function is public, e.g., as in the case of approximating the logsum function, we can just reveal it to the computation peers and outsource just as before. However, when the function is private knowledge to one party, as could be the case for our secure ANN classifier where the design and ordering of activation functions are part of the model owned by $S$, we need to protect this information from the untrusted computation peers. To handle this case, it is important to understand that the approximation parameters (i.e., the coefficients of the polynomials and interval boundaries) are only required to create the garbled selection circuit. We can, however, just as well build a generic garbled selection circuit that takes these inputs form $U$ and $S$ separately in the form of additive shares and recombines them securely within the circuit. On the one hand, this slightly increases the size of the resulting circuit compared to directly inputting the approximation parameters. On the other hand, the computed result is exactly the same and the whole protocol, including the small additional overheads during circuit creation and evaluation, can be outsourced to the cloud.

The last primitive, **EvalProbMass**, is again a different case since it requires both inputs (i.e., the event $X_i = x_i$ and the probability mass function $P(X_i)$) to be known in clear by $U$ and $S$, respectively. Unfortunately, this prevents outsourcing the protocol in its original design as we can provide neither input to the computation peers without violating our security requirements. We could still outsource it using a generic garbled circuit, similar as in outsourcing **FunctionApprox** before. While this would afford for very efficient preprocessing and outsourcing phases, it would render the third phase of outsourced result computation much less efficient than our original design. We thus consider the latter option a last resort for scenarios involving tightly constrained devices. Otherwise, we recommend to move the very efficient original EvalProbMass protocol to the preprocessing phase, e.g., as we already proposed for **Forward** and **Viterbi** with the different goal of batching the many sequential calls to EvalProbMass into one single round.

In summary, all but the **EvalProbMass** primitive can be outsourced by providing the inputs as shares to the computation parties (in the outsourcing phase), who then compute shares of the result mostly according to the original protocols (in the result computation phase), and provide back shares of the results to the peers who can recombine them or continue to compute on them (in the postprocessing phase).

We emphasize two important points for our outsourcing schemes: First, sharing and recombining are very cheap operations and no further preprocessing is required – outsourcing these primitives should be feasible even on constrained mobile devices (we evaluate the exact overheads later). Second, there is no need to involve $U$
or $S$ between the successive execution of multiple outsourceable primitives – the computation peers just keep hold of the shared outputs of the first invoked primitive and subsequently use them as shared input for the invocation of the second primitive, and so on. This argument is the basis for our construction of outsourcing protocols for our five classifiers, Hyperplane, Ann, NaiveBayes, Forward, and Viterbi.

4.5.2.3 Outsourcing our Classifiers

Outsourcing Hyperplane is straightforward, since all underlying primitives, i.e., Rescale, ScalarProduct, and MaxArgmax, can be outsourced as argued before. In the outsourcing phase, the user only needs to create shares $\langle x_i \rangle_{C_U}$ and $\langle x_i \rangle_{C_S}$ of $p(x_i)$ and send them to $C_U$ and $C_S$, respectively. The service $S$ does the same with the weights $w_{\hat{c}_i}$. $C_U$ and $C_S$ then compute Hyperplane as described in Protocol 4.2 (only skipping its initialization steps). This is possible since we compose outsourceable primitives sequentially without the involvement of $U$ or $S$. Finally, instead of recombining the result $\langle c^* \rangle$ by themselves, $C_U$ and $C_S$ provide $\langle c^* \rangle_{C_S}$ and $\langle c^* \rangle_{C_U}$ to $U$ and $S$, who may recombine or reuse these shares as desired.

Outsourcing Ann proceeds as for Hyperplane: When only $U$ outsources computations, the original protocol is carried out between $C_U$ and $S$. When $S$ outsources, too, the outsourcing scheme for FunctionApprox described above is required.

Outsourcing NaiveBayes requires to precompute all required shares $\langle \hat{p}(x_i | c_i) \rangle$ in the preprocessing as the employed EvalProbMass protocol cannot be efficiently outsourced. $U$ and $S$ then add the derived shares locally and directly provide shares of the posteriors $\langle \hat{p}(c_i | \mathcal{F}) \rangle$ to the computation peers who then only compute Argmax and provide back the shared result $\langle c^* \rangle$. While this outsourcing schemes is clearly less efficient in unburdening $U$ and $S$ than the previous two, EvalProbMass causes only very low overheads which are feasible even on mobile devices (cf. Figure A.1 in Appendix A.3). In contrast, NaiveBayes with an underlying Gaussian distribution (Protocol 4.9) can be fully outsourced: $U$ shares $x_i$ and $S$ shares $\mu_i, \sigma_i$ and $1/(−2\sigma_i)$ to $C_U$ and $C_S$ which compute NaiveBayes and Gaussian on these shares and provide back shares of the result.

Outsourcing Forward also requires to precompute all invocations of EvalProbMass to compute shares of the emission scores $\langle \hat{b}_i(o_i) \rangle$. In the outsourcing phase, $U$ then distributes $\langle \hat{b}_i(o_i) \rangle_{C_U}$ to $C_U$, while $S$ provides $\langle \hat{b}_i(o_i) \rangle_{S}$ to $C_S$. $S$ further provides shares $\langle \hat{\pi}_i \rangle$ of the prior state distribution and shares $\langle \hat{\alpha}_i \rangle$ of the transition scores to $C_U$ and $C_S$. Given these shares, $C_U$ and $C_S$ can then compute Forward as specified in Protocol 4.10 (only leaving out the invocation of EvalProbMass).

Outsourcing Viterbi proceeds in the same way as for Forward. The additional backtracking phase cannot be outsourced (as it requires $U$ to know each $s_i^*$ in clear) and must be executed between $U$ and $S$ in the postprocessing phase.

4.5.2.4 Complexity of Outsourcing

We analyze the complexity of the described outsourcing schemes divided into preprocessing and outsourcing. We do not include the result computation phase as it
Table 4.4 Complexity of outsourcing schemes for the different classifiers: We abbreviate EvalProbMass (EPM), providing an additive sharing to both $C_U$ and $C_S$ by $\langle \cdot \rangle_{C_U}$, and sending a single share to one $C_U$ or $C_S$ by $\langle \cdot \rangle_{C_U}$, respectively. The parameters denote the following for Hyperplane, Ann, and NaiveBayes: $n$ the number of features; $m$ the possible values per feature; $k$ the number of classes; $L$ the total number of layers with $m_l$ neurons on layer $1 \leq l \leq L$. For Forward and Viterbi: $N$ the number of states of the HMM; $M$ the size of the emission alphabet; $T$ the length of the observation sequence.

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Preprocessing</th>
<th>Outsourcing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hyperplane</td>
<td>-</td>
<td>$n(\cdot)$</td>
</tr>
<tr>
<td>Ann</td>
<td>$n(\cdot)$</td>
<td>$nk(\cdot)$</td>
</tr>
<tr>
<td>NaiveBayes</td>
<td>$nk\ EPM_m$</td>
<td>$k(\cdot)_{C_U}$</td>
</tr>
<tr>
<td>Forward</td>
<td>$TN\ EPM_M$</td>
<td>$TN(\cdot)_{C_U}$</td>
</tr>
<tr>
<td>Viterbi</td>
<td>$TN\ EPM_M$</td>
<td>$TN(\cdot)_{C_U}, N^2(\cdot)$</td>
</tr>
</tbody>
</table>

The preprocessing phase of NaiveBayes, Forward, and Viterbi involves several invocations of the highly efficient EvalProbMass while Hyperplane and Ann require no preprocessing at all. The outsourcing phase involves for all five approaches only the computation and distribution of additive shares, which is very efficient in terms of processing and communication. Notably, the complexity is much better for the constrained device than for the typically unconstrained service. Based on this brief complexity analysis, we thus expect outsourcing to be feasible for constrained devices. We briefly quantify the actual overheads in the following to complement and validate our complexity analysis.
4. Privacy-preserving Classification and Pattern Recognition

Preprocessing Outsourcing

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>User</th>
<th>Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hyperplane</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Ann</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>NaiveBayes</td>
<td>2.60 ms</td>
<td>0.13 MB</td>
</tr>
<tr>
<td>Forward</td>
<td>1.02 s</td>
<td>113.99 MB</td>
</tr>
<tr>
<td>Viterbi</td>
<td>1.02 s</td>
<td>113.99 MB</td>
</tr>
</tbody>
</table>

Table 4.5: Runtimes and communication for outsourcing the secure classifiers on the largest problem instances considered in our evaluation (cf. Section 4.4.8).

4.5.2.5 Evaluation of Outsourcing

We measure the runtime for computing batches of $10^1$ to $10^7$ additive sharings on an LG Nexus 5 smartphone (2.26 GHz CPU, 16 GB RAM). For the computation of a single additive sharing of $l = 64$ bit, we measure between 0.83 μs ($\sigma = 0.05$ μs) for small batches of size 10 down to 0.57 μs ($\sigma = 0.00$ μs) for large batches of size $10^7$. For $l = 32$ bit, the runtimes are almost exactly half these values. Computing additive sharings is thus very cheap and we find from further measurements on different devices that the actual performance mostly depends on the kernel version, compiler optimizations, and other system internals – the measurements presented above can be seen as a lower bound for reasonably up-to-date smartphones.

Based on the previous measurements, we estimate the time required by $U$ and $S$ to compute the additive shares for outsourcing the different classifiers according to the complexities from Table 4.4. For a defensive estimate, we use measurements on the comparably slow Nexus 5, use batch sizes smaller than the actually required number of sharings, long-term security level of $t = 128$ bit, and assume the largest problem instance considered in our evaluation (Section 4.4.8), i.e., the Human Activity Recognition, MNIST, and Audiology datasets, for Hyperplane, Ann, and NaiveBayes, respectively, as well as the 3HBOH Pfam model for Forward and Viterbi. The overheads for the preprocessing phase are measured directly by running EvalProbMass between $U$ and $S$ on the chosen datasets. We summarize the runtime and communication overheads in Table 4.5.

For Hyperplane and Ann, outsourcing is highly efficient and feasible even on mobile devices and in constrained networks. NaiveBayes incurs small preprocessing overheads, while the outsourcing overheads are clearly negligible. As expected, Forward and Viterbi have exactly the same preprocessing and outsourcing overheads since exactly the same transition and emission scores are shared. Runtimes for preprocessing and outsourcing are feasible on mobile devices, while the communication overhead, especially in the preprocessing phase, might prove challenging to mobile devices that are connected only via a cellular network. However, these overhead estimations consider the largest HMM and observation sequence from our evaluation – overheads for small to medium models range only in the order of kB to a few MB.

Finally, we observe that the outsourcing overheads for the user are always one or two orders of magnitude smaller than for the service provider. This is desired since we expect that users need to outsource more frequently than service providers who usually host their backends in the cloud already.
4.6 Conclusion

4.5.2.6 Summary and Future Work

We proposed secure outsourcing as a second, much more radical strategy to completely relieve mobile users of the high overheads in STC. Our outsourcing schemes allow mobile users and service providers to shift almost all overheads of our secure classifier protocols to a computation cloud. The overheads of the outsourced protocol computation in the cloud are the same as in the original designs and thus clearly feasible in a cloud environment as evaluated in Section 4.4.8. Furthermore, we guarantee security in the semi-honest model, i.e., all inputs, intermediate results, and outputs are protected when the two computation clouds do not collude. With our outsourcing schemes, even users with constrained devices communicating over constrained networks are now able to run our secure classifiers.

4.6 Conclusion

Motivated by the problem field of privacy in machine learning, we identified the need for secure classification services which have received far less attention than the related problem of privacy-preserving training of machine learning models. We addressed this problem in two parts.

First, we developed a framework consisting of several efficient secure building blocks and showed how to compose from them different representative secure classifiers, i.e., linear classifiers, Artificial Neural Networks, Naive Bayes, as well as Forward and Viterbi computation on HMMs. In this part, our main goal was in addressing our second research question (efficiency) and we hence show that our designs realize a significant performance improvement compared to the state-of-the-art in related works. Finally, by validating our designs by applying them to three real-world use cases, spam filtering, bioinformatics, and localization, we identified mobile deployments as important emerging scenarios.

In the second part, we presented two approaches for tailoring our secure classifier designs to constrained environments following up on our observations and discussion of future work in the first part. Focusing mainly on our third research question (customization), we proposed bandwidth-optimized versions of the central (arg)max primitive and highly practical outsourcing schemes for the building blocks and the classifiers in our SHIELD framework. Comparing both strategies, i.e., BOMA and secure outsourcing, we find that the latter is better suited to enable secure classification and pattern recognition in mobile environments but requires two non-colluding computation clouds. BOMA may thus replace outsourcing when the two parties do not have access to a computation cloud, e.g., in ad-hoc networking where two users interact spontaneously in a device-to-device fashion over Bluetooth or WiFi without Internet connectivity [WZCW14]. Furthermore, both strategies are fully compatible and may be combined to complement each other, e.g., when computation clouds aim to trade-off processing against communication as it may be cheaper in large data centers to add computing power than to increase incoming and outgoing network links [PSSZ15]. Concluding, both strategies have a raison d’être.

Based on these observations, we identify outsourcing as the more promising approach to generally address our third research questions, i.e., how to customize STC to chal-
lenged environments. With the main focus of this chapter being on developing the secure classifiers, we only considered the two-party scenario typical for classification and pattern recognition. We have neither considered scenarios where a single data owner wishes to outsource to a single untrusted cloud nor scenarios where more than two parties need to interact. Furthermore, we have focused on processing and bandwidth constraints and not considered potential other constraints. Continuing on our third research question, we thus thoroughly investigate secure outsourcing as a general strategy to tailor secure protocols in a much wider variety of challenged environments in the next chapter.
Outsourced Private Set Intersection

In this chapter, we fully focus on outsourcing the execution of secure protocols, which we have identified in the previous chapter as the most promising way of addressing our third research question, the customization of secure protocols towards scenarios that involve constrained devices or networks. In particular, we concentrate on the Private Set Intersection (PSI) problem in which one, two, or more data owners wish to compute the intersection of their private datasets without the other or any third parties learning those elements that are not contained in the intersection.

We choose PSI for two reasons: First, PSI is an important building block for many applications of Secure Multiparty Computation (SMC) and has been one of the most widely studied SMC problems in the past decade. While numerous efficient PSI protocols have been proposed [FNP04, DCT10, HEK12, PSSZ15, KLS+17], only few consider applications in constrained environments [RCE11, DCFG12]. Second, PSI often needs to be applied to large datasets in real-world applications [PSSZ15, BBDC+11, DCFG12] – the involved overheads especially challenge constrained devices. Based on these two observations, we identify the need to customize existing and design new PSI protocols.

We begin this chapter by motivating PSI and emerging challenges (Section 5.1), then analyze the problem and solution requirements in detail (Section 5.2). Our general approach to tailoring PSI protocols to constrained application scenarios is securely outsourcing storage and computation overheads to more capable peers and networks. As the first main part of this chapter, we design outsourcing schemes for a wide variety of scenarios, i.e., involving a single, two, or multiple data owners who outsource to a single, two, or multiple cloud peers (Section 5.3). The presented schemes were published independently as part of our TraceMixer and BLOOM systems [ZHBW17, ZPH+17], which constitute the second main part of this chapter. TraceMixer [ZHBW17] and BLOOM [ZPH+17] are complete systems for privacy preserving crowd-sensing and genetic testing, respectively. They build on but greatly extend our outsourced PSI protocols and should thus be viewed as independent contributions that we present as two comprehensive case studies for the real-world applicability of our developed outsourcing schemes (Sections 5.4 and 5.5).
5. Outsourced Private Set Intersection

5.1 Motivation

Set intersection is a foundational building block for real-world services—PSI protocols allow mutually untrusting parties to compute intersections between their datasets without revealing them to each other, which is appealing in many scenarios:

**Measuring conversion rates in online advertising:** The success of an advertising campaign is usually measured by the fraction of viewers of an ad that are converted into customers. For online advertisements, conversion rates can be approximated by click rate measurements. Unfortunately, this measure is useless when the final purchase is carried out on a different device or entirely offline, e.g., in mobile advertising, or advertising for brick-and-mortar stores. A more general approach is to compute the intersection between the list of viewers of an ad (held by the ad-network) and the list of customers (held by the advertising merchant). In order to avoid sharing more sensitive data than necessary, both parties can run a PSI protocol to estimate the conversion rate from the intersection of both lists. For example, Facebook estimates with the help of Datalogix\(^1\) the conversion rates for brick-and-mortar stores, utilizing a hashing-based set intersection to reduce the amount of shared private information [Rei12]. However, their protocol was found insecure in [PSSZ15].

**Discovering contacts and presence:** In online social networks and messaging services, it is a common task to first identify other registered users a new user might know and later to determine which contacts are online at a given time or even in close physical proximity. This basically requires computing an intersection between the list of registered users, their online status, or location, as held by the service provider, and the user’s contact list, online status, and location. A simple and widely employed solution is to upload the user’s address book in clear to the service provider and continuously track her online presence and location which causes privacy concerns, e.g., as governmental agencies are increasingly interested in such metadata [GLL14] and may force service providers to grant them access [Lad14]. To protect users’ privacy, contact and relationship discovery [MPGP09, Mar14, PSSZ15], online presence [BDG15], and proximity detection [NTL11] can be run securely using PSI protocols.

**Genomic data analytics:** Cheap whole genome sequencing paves the way towards the large-scale collection and processing of human genome data [Chu05, GBH03], promising among others tremendous advances in biomedical research, more effective diagnoses, and the development of personalized medicine. Many of these applications require finding patients with the same subset of genomic variations or linking genomic databases to correlate genetic disorders that might cause certain diseases. However, collecting, processing, and sharing genome data causes unprecedented privacy risks for the subjects [ACHT13, NAC15, EN14]. This is even aggravated by the common practice to outsource large datasets to cloud services for convenient storage, access, and processing [JZW14, WJT17]. Here, PSI protocols offer an important building block towards privacy-preserving computational genomics [BBDC11, DCFGT12].

---

\(^1\)Datalogix specializes in advertising metrics based on loyalty card data and was acquired by Oracle in December 2014.
Recommender systems: In e-commerce, e-dating, and online social networks a common task is to find people with similar purchase histories, matching interests, or common friends and hobbies in order to recommend further purchases, potential partners, or new friends, respectively – set intersection is clearly a foundational building block of corresponding algorithms. Today, usually all required information is collected and processed centrally by the service provider. However, in many cases the involved information is private, e.g., in the case of location data [Sch10,FS10], or sensitive information can be inferred from it, e.g., the authors of [JM09] show how to infer the sexual orientation of Facebook users. Privacy concerns hence hold back growth of these markets [NTL+11] and leakage of such sensitive data may damages the reputation of service providers [Men16]. In these scenarios, PSI protocols not only protect users’ privacy and unburden service providers from storing critical data. We imagine that PSI can even improve existing recommender systems by allowing the combination of datasets held by different service providers that cannot share data nowadays due to applicable data protection legislation, e.g., a book shop and movie streaming service could together train a collaborative filtering model on the purchase and viewing histories of their combined customers.

Miscellaneous: Further scenarios for PSI are no-fly lists and criminal investigations [Ker12b,DCT10], detecting tax fraud [DCT10], uncovering Botnets [NMH+10], collaborative network security [BD08,PSSZ15], as well as privacy preserving data mining [VC05,LP09]. In all of these examples, two or more parties hold confidential datasets in which they want to identify common elements while minimizing disclosure of the non overlapping parts.

The above scenarios not only motivate PSI in general. We also selected them because they point out different requirements that challenge current PSI protocol designs and motivate outsourced PSI as a solution.

First, PSI must scale to potentially very large datasets: Ad networks like Google’s or Facebook’s reach millions of ad impressions per day that need to be considered when measuring conversion rates. Similarly, a genetics researcher needs to find and correlate common variations within the more than three million base pairs of the human genome. Outsourcing PSI allows harnessing the virtually unlimited storage and computation resources offered by cloud services to ensure the required scalability.

Second, PSI protocols must be able to handle scenarios where data owners are not online at the time of protocol execution. In e-dating, for example, a matching algorithm must sift through the profiles of all registered users, many of which may be offline at that time. Using outsourcing, offline users can apriori delegate storage and processing of their data to a cloud service of their choice which then participates in the PSI protocol on their behalf, e.g., with the operator of the e-dating service.

Third, the parties executing PSI protocols may be mobile users that communicate over networks with intermittent connectivity, limited bandwidth, or high latency. This is the case in the previous example of discovering contacts, presence, and location in mobile applications. As we have showed in the previous chapter, outsourcing incurs little processing and communication overheads such that even mobile users with limited computation, storage, and energy resources can participate.
The dominant line of research in PSI considers almost exclusively the challenge of scalability in the standard Secure Two-Party Computation (STC) setting where two data owners interact directly to compute an intersection of their two datasets [PSZ14, PSSZ15, PSZ16]. Unfortunately, much less attention has been paid to the emerging scenarios that we have just pointed out, i.e., offline or mobile users, single or multiple data owners, or cost metrics other than runtime. Motivated by the importance of PSI and the unsolved arising challenges, we focus on outsourcing schemes for PSI and their applications in this chapter. To this end, we start with a detailed discussion of the problems and challenges of outsourcing in the next section.

5.2 Problem Analysis

We have observed a rising demand for outsourcing PSI in many real-world applications that feature diverse actors and roles. In the following, we abstract from these motivational examples, distill characteristic usage scenarios, and state corresponding requirements for outsourcing PSI. We then analyze existing proposals and finally summarize the main contributions of this chapter.

5.2.1 Problem Statement

We formally define the problem of PSI and Outsourced Private Set Intersection (OPSI) as well as the related problem of computing the intersection cardinality (PSI-CA and OPSI-CA) and compile a list of requirements that ideal OPSI schemes should fulfill.

**Private Set Intersection.** We consider $n$ datasets $D = \{D_1, \ldots, D_n\}$ with elements from a finite universe $U$. Each of the $m \leq n$ users $U_1, \ldots, U_m$, referred to as data owners, holds one or more of these datasets. The data owners want to compute arbitrary intersections between their datasets, i.e., $D_i = \bigcap_{j \in I} D_j$, such that no other party learns anything about the elements $\bigcup_{i=1}^m D_i \setminus D_i$ that are not contained in the intersection except for what is implied by the result $D_i$ and $U_j$'s own datasets. This definition includes the special case where a single data owner $U$ holds all $n$ datasets and no others are involved. This special case covers, e.g., the scenario of a genetics researcher who wants to delegate storage and processing of a genome database to the cloud. We denote a protocol for privately computing the intersection of $|I| = k$ datasets by $k$-PSI.

**Private Set Intersection Cardinality.** We note that many of the motivating examples listed above only require computing the cardinality of the intersection, e.g., in genetic testing [BBDC+11], association rule mining [VC05], location sharing and discovery of proximity [NTL+11]. To increase privacy, it is even desired to explicitly hide the particular elements in the intersection whenever possible. We refer to this problem as PSI-CA, which is evidently closely related to PSI and thus also treated in this chapter. Formally, PSI-CA is defined analogously to PSI, i.e., given $n$ datasets $D = \{D_1, \ldots, D_n\}$, the users $U_1, \ldots, U_m$ want to compute the cardinality $|D_i| = |\bigcap_{j \in I} D_j|$ of the intersection of datasets $D_{i \in I} \in D$ without $U_j$ learning anything about the original datasets other than what is implied by the result and $U_j$’s input. We denote any protocol that securely computes the desired cardinality of the intersection of $k$ datasets as $k$-PSI-CA.
5.2 Problem Analysis

Unconstrained cloud environment

Constrained devices and networks

User U1

1) Preprocessing

Cloud C1

Cloud C2

... Cloud Cn

3) Results computation

2) Setup

User U2

4) Postprocessing

User Un

Trust spheres

Figure 5.1 Overview of our outsourcing scenario: The n data owners U1, ..., Un hold n datasets D1, ..., Dn that they want to outsource to m cloud peers of their choice in order to delegate computation of intersections of these datasets. The primary goal of any outsourcing scheme is to minimize overheads for the data owners (i.e., preprocessing, setup, and postprocessing phases) while keeping cloud overheads for result computation feasible.

Outsourced Private Set Intersection (Cardinality). As illustrated in Figure 5.1, we define our outsourcing model as a generalization of the model proposed for outsourcing our SHEILD framework (Section 4.5.2). At the start, the data owners U1, ..., Um may engage in a first preprocessing phase after which each Ui individually chooses an outsourcing peer Ci and announces its choice to the other parties – two data owners may independently choose the same cloud service thus we end up with o \leq m cloud peers. Each data owner Ui then sets up the clouds C1, ..., Co with the required data for the subsequent results computation phase in which the clouds compute the desired set intersections. Finally, the clouds return the protected results to the original parties which may postprocess, decrypt, or use them in subsequent (secure) computations. We denote by k-OPSI(\(m^o\)) any outsourcing scheme that allows compute an intersection of k \leq n datasets held by a subset of m data owners in an untrusted network of o \leq m cloud services. Analogous, we denote any outsourcing scheme that computes the cardinality of such intersections by k-OPSI-CA(\(m^o\)).

Reviewing our motivating examples, we observe that the need for (outsourcing) PSI and PSI-CA arises in many different scenarios from which we deduce the following requirements. A subset of these goals has also been recognized in [ATD15,ABB16].

Feasible overheads: The overheads for all protocol steps that involve the data owners U1, ..., Un should be minimized, i.e., the preprocessing, outsourcing, and postprocessing phases. Depending on the application scenario, these overheads should be feasible for mobile devices that use constrained networks, e.g., cellular networks with limited bandwidth and high latency. At the same time, the overheads for the outsourced result computation must remain feasible for the cloud peers C1, ..., Co and should not significantly increase compared to an execution of the native PSI protocol directly between the data owners.

Multiple data owners: We observe that the need for outsourcing PSI arises in scenarios with a single data owner (e.g., in genomics [LYS15,TJW‘16] or delegated computation [BFR13]), in the standard two-party setting (e.g., in on-
line advertising [PSSZ15], discovery of common contacts [MPGP09, NTL+11, BDG15], or checking no-fly lists [Ker12b, DCT10]), and in scenarios with multiple data owners (e.g., recommender systems and data mining [VC05, LP09], Botnet detection [NMH+10], or collaborative network security [PSSZ15]). OPSI schemes must thus support variable numbers of data owners.

**Repeated independent execution:** Once the data owners have outsourced their datasets, it should be possible to compute arbitrary and unlimited numbers of intersections on these datasets afterwards without requiring any further involvement of the data owners, e.g., to download and re-encode datasets. In particular, the preprocessing and setup phases must be independent of the number of intersections computed in the results computation phase and these phases should not overlap. For example, a genetics researcher must query a database of human genomes for a large number of different genetic variations and it would be impractical to download, re-encode, and upload the database for each new query.

**Composability:** OPSI protocols should be composable in the sense that the computed intersection constitutes a new outsourced set in itself that can later be intersected again with other datasets without requiring involvement of the data owners, e.g., to download, decrypt, re-encode, and upload the intersection. Composability is desirable as most research focuses on 2-PSI protocols which must be strung together to compute intersections of \( k > 2 \) sets. In its most general form, composability means that the OPSI scheme can be freely used as a building block in a surrounding secure computation protocol, implying that the OPSI scheme must handle the case when all inputs, intermediate values, and outputs are encrypted.

**Flexible data handling:** Data owners should be able to outsource their datasets at any time and to modify, add, or delete data later without the need to download and fully re-encode already outsourced datasets. E.g., in mobile messaging applications, online social networks, or e-dating platforms, users do not join all at once but dynamically over time and thus need to repeatedly update location information as well as hobbies and interests.

**Security and trust model:** We require that outsourcing schemes are secure in the semi-honest adversary model. As defined in detail in Section 2.2, a semi-honest attacker correctly follows the protocol but tries to infer additional information from the protocol transcript. This model is reasonable in the outsourcing context where cloud providers must preserve their reputation and thus have a strong interest in executing the computation correctly [KMRS14, PSSZ15, ATD15]. Compared to security in the malicious model, the semi-honest model allows for much more efficient protocols while still protecting against insiders and outsiders that try to learn the private datasets and intersections.

Importantly, semi-honest behavior does not preclude collusion [LP00], e.g., between data owners and cloud peers. It is thus an important decision whether to allow each of the \( m \) data owners to choose its own cloud peer (which she trusts not to collude with the others) or to require data owners to agree on some common \( o \ll m \) cloud peers (that all data owners need to trust not to collude). We note that both options and their respective advantages have
been considered in the literature, e.g., outsourcing to a smaller number of peers
usually allows faster protocols (as complexity is often quadratic in the number
of peers) but is more vulnerable to collusion than outsourcing to many peers.
The general approach is thus to prove a threshold $\tau$ of colluding peers against
which the protocol remains secure. We point out that the case of a single data
owner outsourcing to a single cloud peer presents a notable exception in the
context of collusion: The cloud peer could only collude with the data owner
itself which evidently does not present a reasonable attack scenario as the data
owner would only learn information she either already knows (the dataset) or
learns from the computation (the intersection result).

We conclude that we require the following three types of outsourcing schemes for PSI:
i) $k$-OPSI($^m_1$), referred to as single-party outsourcing, ii) $2$-OPSI($^2_2$), correspond-
ing to outsourcing in the standard two-party setting (also assumed in the previous
chapter), and iii) $k$-OPSI($^m_0$), the most general case in our analysis above. Besides
contributing corresponding OPsi schemes, we also design further variants that are
more efficient or general, i.e., $k$-OPSI($^m_0$) (multiple data owners outsourcing to two
cloud peers) and $m$-times parallel $2$-OPSI($^1_1$) (a single data owners outsourcing to
a single cloud peer).

5.2.2 Related Work

We analyze related work with respect to our problem definition and requirements.
Beginning with a short overview of PSI protocols in the standard (non-outsourcing)
setting, we then focus on existing approaches to outsourcing PSI, and finally briefly
discuss orthogonal work on delegated computing outside the scope of the PSI setting.
Although there are few special purpose protocols [VC05, DCGT12], the majority of
PSI-CA protocols are derived from PSI protocols and share the same basic concepts
and properties. In particular, this holds for the currently most efficient PSI and
PSI-CA protocols. We thus do not present a separate discussion of PSI-CA schemes.

5.2.2.1 Standard PSI protocols

Due to the abundance of proposed PSI protocols, we provide only a brief summary of
the most important techniques and refer to [PSSZ15, BA16] for an in-depth discus-

Naive Hashing. Set intersection protocols based on naive hashing proceed roughly
as follows: Having agreed on a common salt $\kappa$, the data owners $U^i$ compute the salted
hash of their elements $H(x_{i,j} \oplus \kappa)$, randomly shuffle them under a secret permutation
$\pi_i$, and finally share the resulting hashed and permuted elements $H(x_{i,\pi_i(j)} \oplus \kappa)$ with
the others in clear. Each party can then compute the intersection (cardinality)
locally on the permuted hashes and recover the original elements by hashing the
own dataset again and comparing to the hashes in the intersection. Set intersection
protocols based on naive hashing are very efficient and easy to implement. They
are thus widely used in real-world systems, e.g., to measure ad conversion rates of Facebook ads [Rei12] or for contact discovery in the messenger Signal [Mar14].

The naive hashing approach can also be easily turned into an OPSI scheme that affords all our stated goals except security: When elements come from a small universe $U$ it is feasible to recover all original elements in the intersected sets by brute force, i.e., a semi-honest $U_j$ or cloud peer $C_j$ checks whether $H(x \oplus \kappa) = x_{i,j}$ for all $x \in U$ and $x_{i,j} \in D_i$ to recover party $U_i$’s dataset $D_i$. As noted in the context of common contact discovery in the Signal messenger [Mar14], this threat exists in real-world systems and motivates the need for PSI. Also noted in [Mar14], existing PSI protocols are not efficient enough for mobile devices which further motivates outsourcing.

Server-aided PSI. Different PSI protocols have been proposed that use an untrusted third party with the goal to increase efficiency. Server-aided PSI thus has some shared characteristics with outsourcing but usually involves data owners to a much higher degree. In a recent and very efficient proposal [KMRS14], the authors secure the previously presented naive hashing approach using a third party server that must not collude with any of the data owners. However, the protocols in [KMRS14] are one-off, i.e., data owners need to interact with the third party for each new intersection which violates our repeatability and composability requirements. In three of the four presented protocols the server also learns the size of the intersection which violates our security model and renders them inapplicable as PSI-CA protocols.

PSI based on PKC. Among the first proposals are PSI protocols based on PKC and they have since had a long tradition, e.g., basing on Diffie-Hellman keyexchange [Mea86,FFH99], ElGamal [FNP04,FFNP14], or Rivest-Shamir-Adleman (RSA) [DCT10,DCT12]. Dedicated PSI-CA protocols were proposed based on Diffie-Hellman [DCGT12] or commutative keyed one way hash function [VC05]. Most of these protocols are simple to implement but involve expensive modular exponentiations and cannot be outsourced as plaintext knowledge of the dataset is required. The results presented in [PSSZ15] show that PSI based on PKC is slower than other state of the art protocols. We thus do not further consider this class of PSI protocols.

PSI based on generic SMC. In Section 2.3, we have described four different protocols that securely evaluate any Boolean or arithmetic circuit. Hence, PSI can be solved by encoding traditional intersection algorithms in such circuits and executing them securely using the described generic secure protocols. The main challenge is then to design efficient circuits.

In an effort to answer whether special purpose PSI protocols are better than those over generic circuits, Huang et al. [HEK12] propose three different circuits for 2-PSI, i.e., Bitwise-AND (BWA), Pairwise-Comparisons (PWC), and Sort-Compare-\-Shuffle (SCS). The authors show that their designs compare favorably against special purpose PKC-based protocols. BWA and PWC target small datasets with tens to hundreds of elements while the third circuit, SCS, scales to larger sets with thousands of elements. In the SCS approach, the two parties locally sort their datasets, then obliviously merge them into a single set using Garbled Circuits (GCs), extract adjacent elements that are equal (these are the intersecting elements), and finally shuffle the result to prevent information leakage. Importantly, the authors point out that PSI over generic circuits has the advantage of enabling the embedding of PSI into more complex secure computations (which we termed composability). As an
example, the authors demonstrate how to add an auditing mechanism that reveals intersections only when they are below a certain size, i.e., they essentially build an PSI-CA scheme from their protocol.

Acknowledging the importance of generic circuits for PSI, Pinkas et al. [PSSZ15] present a novel 2-PSI protocol, Circuit-phasing, whose underlying circuit is roughly 3x smaller compared to the SCS circuit of [HEK12] and is designed for Single-Instruction-Multiple-Data (SIMD). To the best of our knowledge, this is currently the most efficient generic approach and computes intersections of sets with up to $2^{16}$ elements in tens of seconds. On the downside, the used hashing techniques prevent extending their scheme to a general k-PSI protocol without losing much of its efficiency. Still, we are able to obtain very efficient 2-OPSI$\binom{m}{2}$ and 2-OPSI-CA$\binom{m}{2}$ outsourcing schemes from this approach (Section 5.3.2).

PSI based on OT. The most recent class of PSI protocols is based on OT, motivated by the tremendous efficiency improvements of OT extension protocols [ALSZ13, ALSZ15]. This line of research was initiated by the proposal in [DCW13], which uses OT to compute set intersection between a standard and a garbled Bloom filter\footnote{The notion of garbling is different in [DCW13] than for Yao’s GC protocol and actually has more resemblance with secret sharing: To add an element into a garbled Bloom filter, it is first secret shared and the shares are stored in the Bloom filter.}. Further improved OT-based PSI protocols were proposed in [PSZ14, PSSZ15, PSZ16] that combine very efficient special flavors of OT extensions (OT over random strings) with advanced hashing techniques. While OT-based protocols are currently the most efficient PSI protocols, they cannot be outsourced since the OT runs (which are the only interactive steps between the two parties) require plaintext knowledge of the elements (we discussed this problem before in Section 4.5.2.2 where we tried to outsource EvalProbMass). With respect to our goal of building outsourcing schemes for PSI, these approaches are thus irrelevant given that there is currently no outsourcing scheme for OT and OT extensions.

All of the designs so far have been designed and optimized for efficiency – none of them explicitly considers outsourcing storage and processing of datasets to one or multiple untrusted clouds. We now shift our focus to protocols designed specifically for outsourcing and analyze them with respect to our requirements for OPSI.

5.2.2.2 Outsourced PSI protocols

One of the first OPSI schemes was proposed by Kerschbaum [Ker12b] and allows two data owners $U_0$ and $U_1$ to outsource set intersection to a single untrusted cloud, i.e., 2-OPSI$\binom{2}{1}$. The scheme represents sets as Bloom filters and uses the partially homomorphic Boneh-Go-Nissim encryption scheme [BGN05] combined with the Sander-Young-Yung technique [SYY99] to securely intersect them in the cloud by computing pairwise logical ANDs of the Bloom filters’ bits. The scheme affords repeatability (with only slight modifications) and the bitwise encryption of the Bloom filter allows flexibly adding and modifying datasets. However, the employed cryptosystems only permit circuit with at most two sequential logical AND gates. This approach is thus inapplicable to intersections of $k > 2$ datasets and is not composable. Furthermore, it requires the expensive PKC encryption of one Bloom filter.
per element in the second party’s set and is thus unlikely to scale to larger sets (the author presents no performance evaluation of the proposed OPSI scheme to prove otherwise). Finally, it is unclear whether Kerschbaum’s OPSI could be turned into an OPSI-CA scheme: Our idea (presented in Section 5.3.3) of adding the bits of the intersection Bloom filter under encryption to obtain the cardinality does not apply here since the encryption schemes employed in [Ker12b] do not support addition.

A second OPSI protocol in a slightly different setting was proposed by Kerschbaum in [Ker12a]. This protocol is one-off, i.e., it requires interaction with the data owner for each new intersection, and thus neither achieves repeatability nor composability.

The protocols proposed by Liu et al. [LNZ+14] and Zheng et al. [ZX15] are based on PKC and allow two clients to outsource set intersection to a single cloud (2-OPSI). Both schemes can be extended to the intersection of multiple sets (k-OPSI) but do not offer the important repeatability or composability properties for two reasons: First, besides always revealing the size of the intersection to the untrusted server and thus rendering them insecure and inapplicable to OPSI-CA, subsequent intersections reveal further non-trivial information about the intersected sets to the cloud, as argued in [ATD15]. Second, the data owners must re-prepare datasets for each new intersection. Finally, Liu et al.’s protocol [LNZ+14] has quadratic complexity in the size of the datasets which suggests prohibitive overheads (that are not evaluated) while the scheme by Zheng et al. [ZX15] has linear overheads but still requires minutes to hours for the intersections of two sets with 2^{14} elements – as a comparison, remember that the generic scheme of [PSSZ15] requires only tens of seconds to intersect two sets with 2^{16} elements.

Abadi et al. [ATD15] use the well-established technique of representing sets as polynomials [FNP04,FHN14,Haz15] and use Partially Homomorphic Encryption (PHE) to build a 2-OPSI scheme (also sketching a generalization to k-OPSI). However, they do not evaluate the performance of their protocol and the authors of [PSSZ14,PSSZ15,PSZ16] find that set intersection based on oblivious polynomial evaluation is performance-wise inferior to other approaches. Furthermore, each run of the protocol involves non-trivial interaction between clients (i.e., one of the client has to compute n modular exponentiations), which violates our repeatability requirement. Finally, Abadi et al.’s protocol is not composable since the encrypted result cannot subsequently be used in another run of the protocol or another secure computation and it remains unclear how to extend it to OPSI-CA.

The most recent proposal is due to Blanton and Aguilar [BA16] who propose outsourced set intersection by implementing the sort-compare-shuffle approach by Huang et al. [HEK12] as an arithmetic circuit that they securely evaluate in the multi-party setting using Shamir’s secret sharing, yielding 2-OPSI and 2-OPSI-CA schemes. Theirs is the only approach that completely fulfills our repeatability and composability requirements and even incurs only very small overheads on the data owners. Overall, the contribution of [BA16] is rather in providing secure protocols in the information theoretical model for semi-honest and malicious adversaries and less focus is on the efficiency of these protocols. For example, the authors report runtimes of 25s for the intersection of two small sets of 2^{11} elements even in a small network of only three cloud peers that are connected over Gigabit LAN – as a comparison, the circuit-phasing protocol in [PSSZ15] has a similar runtime on much larger sets of 2^{18} elements in a comparable setting.
5.2. Problem Analysis

5.2.2.3 Orthogonal works

In a different line of research, Papamanthou et al. [PTT11], Backes et al. [BFR13], and Canetti et al. [CPPT14] propose outsourcing different computations, among them set operations, to the cloud. With the different goal of providing verifiability of the outsourced computation, the authors do not consider the privacy of the data owners, i.e., datasets are outsourced as plaintexts and protocols do not to work over encrypted data to protect sensitive inputs from an untrusted cloud. These approaches hence do not comply with our security model but could potentially be combined if verifiability is required.

Fiore et al. [FGP14] construct a scheme that allows delegating computation of linear combinations and polynomials over encrypted data to an untrusted cloud and to verify the correctness of the computation. They do not specifically consider set intersection as an application but it could be implemented on top of their framework. However, the authors report prohibitive runtimes of hundreds of seconds for computing a simple scalar product on vectors of 1000 elements – remember that the \textsc{ScalarProduct} protocol in our SHIELD framework (Protocol 4.3, Section 4.4.2.1 required only few milliseconds for the same task as evaluated in Appendix A.1). Since PSI protocols are more complex than scalar products, we expect at least these overheads when running, e.g., a Bloom filter-based PSI protocol on top of Fiore et al.’s approach.

5.2.2.4 Summary

The problem of computing set intersections and their cardinalities privately has been extensively studied and highly efficient protocols have been proposed that scale to sets with millions of elements. Existing proposals provide valuable insights into the design of efficient PSI and PSI-CA protocols but, when viewed from an outsourcing perspective, come with one or more of the three following disadvantages: First, they are usually designed for the two-party setting, making it difficult, inefficient, and sometimes impossible to securely compose them to the more general k-PSI or use them as building blocks in a more complex secure computation. Second, most protocols are interactive and require the data owners to actively participate in each protocol execution which is undesirable for the design of outsourcing schemes. Third, the security models assume two or more non-colluding parties, rendering these approaches inapplicable to the single-party outsourcing setting.

With the advent of cloud computing and emerging application scenarios for secure protocols in mobile settings, the interest in outsourcing PSI, or outsourced secure computations in general, has picked up. However, none of the proposals fulfills our six requirements for ideal PSI and PSI-CA schemes, i.e., i) scalability to large datasets, ii) support for multiple data owners, iii) repeatability of intersections, iv) composability with other set operations or secure computations, v) flexibility to add, delete, and modify outsourced data, and vi) security in the semi-honest model.
5.2.3 Our Contributions

Motivated by the wide applicability of PSI, the great progress made in the traditional setting, and the shortcomings of existing proposals in the outsourcing setting, we set out to design efficient and secure OPSI schemes to overcome the challenges posed by, e.g., large datasets, dynamically joining and leaving users, or constrained devices and networks. Our general approach is to turn existing efficient PSI protocols into OPSI schemes where possible and only to develop novel protocols when required. As a result, we propose outsourcing schemes for several scenarios that each realize different trade-offs between generality, security (especially collusion resistance), and efficiency. In order to validate our results and to demonstrate their real-world applicability, we present two comprehensive case studies for our approaches. Our main contributions are the following:

**Outsourced PSI and PSI-CA:** We present a suite of OPSI and OPSI-CA schemes for a comprehensive range of settings, i.e., secure multi-party, two-party, and single-party outsourcing: i) a k-OPSI($m^k$) scheme based on the k-PSI protocol by Many et al. [MBD12], ii) a k-OPSI($m^2$) scheme from the circuit-based 2-PSI protocol due to Pinkas et al. [PSSZ15], and iii) a novel k-OPSI-CA($m^2$) scheme together with a second significantly optimized version for the special case of 2-OPSI-CA($m^2$). These schemes were developed as part of the TraceMixer [ZHBW17] and BLOOM [ZPH+17] systems that are independent contributions and presented as case studies in this chapter to validate and demonstrate the real-world applicability of our PSI protocols.

**Case Study of Privacy-preserving Crowd-Sensing:** We propose the TraceMixer system [ZHBW17], a novel location privacy protection mechanism that combines our 2-OPSI-CA($m^2$) scheme with further orthogonal data anonymization techniques to allow the privacy-preserving collection of location traces in a crowd-sensing setting. We deploy TraceMixer as privacy mechanism in a real-world crowd-sensing campaign for the creation of high precision elevation profiles using barometric altitude information on commercial off-the-shelf smart phones. Thus, TraceMixer not only demonstrates the real-world applicability of the developed 2-OPSI-CA($m^2$) scheme but should be viewed as an independent contribution beyond the scope of secure outsourcing.

**Case Study of Privacy-preserving Genetic Testing:** We propose BLOOM [ZPH+17], a system for outsourced genetic testing based on our novel k-OPSI-CA($m^k$) and 2-OPSI-CA($m^2$) schemes, which ranked runner-up in the 2016 edition of the Secure Genome Analysis competition [iDa16,WJT+17,HTJ16] posed by the center for integrating Data for Analysis, Anonymization, and SHaring (iDASH)\(^\text{3}\). We show that our novel proposal for k-OPSI-CA($m^k$), FHE-BLOOM, scales to sets with millions of elements while our second scheme, PHE-BLOOM, improves runtimes by three to four orders of magnitude for the special case of 2-OPSI-CA($m^2$).

\(^3\)iDASH is a National Center for Biomedical Computing under the Roadmap for Bioinformatics and Computational Biology established by the National Institutes for Health in the United States.
5.3 Outsourced Private Set Intersection

We now present OPSI and OPSI-CA protocols for a comprehensive range of settings. We categorize the proposed schemes by their trust model, i.e., how many cloud peers computations are outsourced to. In particular, we present schemes for i) outsourcing to multiple cloud peers (Section 5.3.1), ii) outsourcing to two peers (Section 5.3.2), and iii) outsourcing to a single cloud peer (Section 5.3.3). We evaluate the proposed schemes and discuss the achieved properties afterwards (Section 5.3.4).

Categorization by trust models is reasonable since each enables another class of secure protocols with vastly different properties: Outsourcing to many peers realizes the most flexible trust model as it allows each data owner to individually choose a cloud peer but requires multi-party protocols that are often slower than GCS or Goldreich-Micali-Wigderson (GMW), the prominent two-party approaches. Outsourcing to two peers allows capitalizing on the strong line of research in two-party PSI protocols but requires multiple data owners to agree on two cloud peers that they trust not to collude. Outsourcing to a single peer is the most secure solution for a single data owner and additionally has the advantage that the outsourced computations are immune to network latency. All three considered settings thus have their raison d’être and will be treated in this section.

We begin with the presentation of a scheme for most general and at the same time most inefficient case of $k$-OPSI($m$) (Section 5.3.1). Reducing the number of cloud peers to two (Section 5.3.2) and even to a single cloud (Section 5.3.3) then allows us to use different cryptographic tools for the design of more efficient protocols.

5.3.1 Outsourcing to Many Peers

We turn the multiparty PSI protocol due to Many et al. [MBD12] into a $k$-OPSI($m$) scheme, i.e., we allow $m$ peers to outsource the intersection of $k \leq m$ of their sets to $o \leq m$ cloud peers. The proposal of [MBD12] as well as our novel protocols for $k$-OPSI($1$) (Section 5.3.3) are based on Bloom filters. We thus provide a brief background first and then present the outsourcing scheme.

Background on Bloom filters

Bloom filters [Blo70] offer a space-efficient probabilistic data structure to represent sets. They are particularly efficient when checking set membership which is a central part of any set intersection protocol. Formally, an empty Bloom filter is a bit array $B \in \{0,1\}^l$ of length $l$ with all bits $b_i \in B$ initialized to zero. We denote $B \leftarrow \text{CreateBF}(l)$ the creation of such an empty Bloom filter of length $l$.

Before inserting elements to the Bloom filter, we need to fix $k$ distinct hash functions $H_1, \ldots, H_k : \mathcal{U} \rightarrow \{0,\ldots,l-1\}$ that map from the universe of elements $\mathcal{U}$ into the Bloom filter $B$. To add an element $x \in \mathcal{U}$ to $B$, we compute positions $i_1 = H_1(x), \ldots, i_k = H_k(x)$ and set the corresponding bits $b_{i_1}, \ldots, b_{i_k}$ to one. This operation is referred to as $B.add(x)$ and we write $B = \text{CreateBF}(l, D)$ as a short hand for creating an empty Bloom filter and successively adding all elements $x \in D$ to it.
5. Outsourced Private Set Intersection

Input: Each data owner $U_i$ holds $D_i = \{x_{i,1}, \ldots, x_{i,n}\}$
Output: Intersection $D_i = \bigcap_{j=1}^{m} D_j$ and estimated cardinality $|D_i|$

Preprocessing:
$U_i \Rightarrow B_i \leftarrow \text{CreateBF}(l, D_i)$
$U_i \Rightarrow U_1, \ldots, U_m : \text{Choose and announce } C_i$

Setup:
$U_i \Rightarrow C_1, \ldots, C_m : (B_i) = (\langle b_{i,0}, \ldots, |b_{i,t-1} \rangle) \leftrightarrow \text{SHARE}(B_i)$

Results computation:
$C_i \leftrightarrow C_j : \langle b_{i,j} \rangle = \langle b_{i,0} \rangle \oplus \cdots \oplus \langle b_{i,n} \rangle, \forall j = 1, \ldots, l$
$C_i : \langle |D_i|' \rangle = \langle |b_{i,1} \rangle \oplus \cdots \oplus \langle |b_{i,t} \rangle \rangle$

Postprocessing:
$U_i \leftrightarrow C_1, \ldots, C_m : (\langle |D_i|' \rangle, (B_i) = (\langle b_{i,0}, \ldots, |b_{i,t-1} \rangle))$
$U_i : \langle |D_i|'/k \rangle \leftrightarrow \text{RECOMBINE}(\langle |D_i|' \rangle), B_i \leftarrow \text{RECOMBINE}(\langle B_i \rangle)$
$U_i : |D_i| = \langle |D_i|'/k \rangle$
$U_i : D_i = \{x_{i,j} \in D_i \land B_i.\text{contains}(x_{i,j})\}$

Protocol 5.1 Combined k-OPSI($m$) and k-OPSI-CA($m$) schemes based on the k-PSI scheme due to Many et al. [MBD12]: We adopt the original protocol to our outsourcing setting and extend it with functionality for efficiently computing the set intersection cardinality.

To test set membership $x \in B$, denoted by $BF.\text{contains}(x)$, we check whether the bits at all $k$ positions $H_1(x), \ldots, H_k(x)$ are set. Due to hash collisions, this test can produce false positives (but not false negatives). The probability $p = (1 - (1 - 1/k)^{kn})^k$ for a false positive is determined by the number of hash functions $k$, the number of added elements $n$, and the length $l$ of the Bloom filter. For a fixed $n$, $p$ is minimized by setting $l = -n \log(p)/\log(2)$ and $k = -\log(p)/\log(2)$.

From k-PSI to k-OPSI($m$) and k-OPSI-CA($m$)

Protocol 5.1 shows our combined k-OPSI($m$) and k-OPSI-CA($m$) scheme adopted from the Bloom filter-based k-PSI protocol of [MBD12] and complemented by our mechanism for computing the set cardinality on Bloom filters [ZPH+17].

In the preprocessing phase, each data owner $U_i$ encodes her dataset $D_i$ in a Bloom filter $B_i$ of fixed length $l$ using hash functions $H_1, \ldots, H_k$. $U_i$ then chooses a cloud peer $C_i$, which she trusts not to collude with the other cloud peers or data owners, and announces her choice. The resulting network has $o \leq m$ cloud peers, when two or more data owners (coincidentally) choose the same cloud.

In the setup phase, each $U_i$ shares her Bloom filter $B_i$ bitwise to the cloud peers using an $(t,n)$-out-of-$o$ linear secret sharing scheme, e.g., Shamir’s secret sharing scheme. Note that $(t,n)$-secret sharing provides protection against up to $\tau < t$ colluding peers and multiplication on shares limits the threshold by $t \leq [(o+1)/2]$ in the semi-honest model (cf. Section 2.3.3.1).

In the results computation phase, only the cloud peers interact in order to multiply the shared Bloom filters $B_i$ at each position $j = 1.\ldots l$ to obtain the bitwise shared Bloom filter $B_C$ encoding the desired set intersection of $D_1, \ldots, D_m$. Note that it is trivial to only compute the intersection on any subset of $k \leq m$ of these datasets. To
obtain a share of the cardinality of the intersection, the cloud peers locally sum the bits of the intersection Bloom filter which estimates the size of the set intersection.

Finally, the cloud peers return their shares \( \langle B \cap \rangle \) of the intersection Bloom filter to the data owners who can locally recombine \( B \cap \). To obtain the actual intersection \( D_1 \cap D_2 = \bigcap D_i \) for each of the elements of its own set \( D_i \) since there is no direct way to obtain from a Bloom filter the elements that it contains. Using this approach, we have to bear in mind that Bloom filters can produce false positives due to hash collisions with probability \( p \) that can be calibrated by the number of hash functions \( k \) and the length of the Bloom filter \( l \) as detailed above. If the data owners should only learn the cardinality of the intersection, the cloud peers only return the shares \( \langle |D_1 \cap| \rangle \) of the estimated intersection cardinality. Data owners need to scale down this value by the number of hash functions \( k \). Note that an accumulation of hash collisions leads to a slight underestimation of the cardinality of the set intersection that we quantify in detail in Section 5.3.4.1.

### 5.3.2 Outsourcing to Two Peers

Most existing PSI protocols are set in the two-party setting which allows employing other cryptographic protocols than in the multiparty setting. It is thus natural to examine whether we can build more efficient OPSI protocols from these works that outsource to only two cloud peers. We thus show how to turn the circuit phasing PSI protocol due to Pinkas et al. [PSSZ15] into a 2-OPSI\(^{(m)}\) scheme and then how to extend this to \( k\)-OPSI\(^{(m)}\). The 2-OPSI\(^{(m)}\) scheme was published as part of our TraceMixer system [ZHBW17] for location privacy in crowd-sensing which we present as our first case study in Section 5.4.

#### 2-PSI due to Pinkas et al. [PSSZ15]

We briefly explain the original 2-PSI protocol from [PSSZ15]: Each of the two data owners, \( U_1 \) and \( U_2 \), holds in clear one of the two sets \( D_1 \) and \( D_2 \) to be intersected. Each peer represents its respective set as a hash table with a configurable amount of \( \beta \in \mathcal{O}(n) \) bins, \( U_1 \) using simple hashing and \( U_2 \) using Cuckoo hashing [PR01]. To handle hash collisions in simple hashing, each bin has \( \max_{\beta} \in \mathcal{O}(\log(n)) \) slots, while bins have only a single slot in Cuckoo hashing and collisions are stored on a stash of a small size \( s \in \mathcal{O}(1/\log(n)) \). \( U_1 \) and \( U_2 \) pad empty bins and slots in their hash tables and the stash with distinct items \( d_1 \neq d_2 \) in order to prevent information leakage. The two data owners then securely evaluate (i.e., using GC or GMW) in parallel a batch of \( \beta \cdot \max_{\beta} + s \cdot n \) comparison circuits that check each element in the bins of \( U_1 \) to those of \( U_2 \) in the same bin for equality as well as each of \( U_1 \)'s \( n \) set elements to the \( s \) elements on the stash of \( U_2 \). The output are \( \beta + s \) single bits returned to \( U_2 \) that indicate at which positions intersections occurred. \( U_2 \) thus needs to keep a mapping of set elements to hash table positions in order to recover the original elements of the intersection.
From 2-PSI to 2-OPSI\((m)\) and 2-OPSI-CA\((m)\)

Our basic idea for turning the described 2-PSI protocol into a 2-OPSI\((m)\) scheme is for data owners to carry out the hashing steps of the original protocol themselves and provide cloud peers the hash tables as Boolean shares. While these shares do not expose any information to a single cloud peer, they allow both parties to execute the main part of the protocol together (i.e., the secure evaluation of the batch of comparison circuits) either directly with the GMW approach or by converting shares into garbled values in order to evaluate the respective circuits using Yao’s GC protocol. Protocol 5.2 provides a high-level overview of the required steps divided into the four phases of our outsourcing model. We refer to the original work by Pinkas et al. [PSSZ15] for all details on the hashing techniques and parameter selection.

Different to the original protocol, in our outsourcing scheme each dataset is hashed both using simple and cuckoo hashing in order to provide repeatability and support for multiple data owners. Storing both formats is required because of the asymmetry in the original protocol, i.e., \(U_1\) uses simple hashing and \(U_2\) uses cuckoo hashing. To compute \(D_1 \cap D_2\), cloud peers operate over hash tables \(H_{G1}\) and \(H_{C2}\), while computing \(D_2 \cap D_1\) requires the opposite hash tables \(H_{G2}\) and \(H_{C1}\). Outsourcing both representations for each dataset allows intersecting arbitrary pairs of the \(m\) datasets outsourced by the \(m\) data owners – without this modification we could only intersect the datasets in the simple hashing representation with those in the Cuckoo hashing format.
5.3. Outsourced Private Set Intersection

In the results computation phase, the two cloud peers compute the same batch of $\beta \cdot \text{max}_\beta + sn$ comparisons between the two hash tables as in the original protocol. We left out the comparisons for the elements on the stash in Protocol 5.2 to simplify the presentation (the stash is treated analogously to the hash table bins). Compared to the original protocol, we contribute additional functionality to efficiently compute the set intersection securely within the circuit. To this end, we use the shared comparison bits to compute the set intersection cardinality which can be efficiently computed in a circuit of approximately $(\beta + s)\log_2(\beta + s)$ non-linear gates by using addition circuits arranged in a tree with $\lceil \log_2(\beta + s) \rceil$ levels where the adders’ bitlengths increase by one on each level.

In the postprocessing phase, the data owners download and recombine the single share $\langle |D_1 \cap D_2| \rangle$ to learn the size of the set intersection. If they also want to learn the actual set intersection $D_1 \cap D_2$, they additionally need to download and recombine the comparison bits $B_{S, \cap}$ and $B_{C, \cap}$ corresponding to the structure of the simple and Cuckoo hashing tables and combine these with their original datasets $D_i$ to recover the original elements in the intersection $D_1 \cap D_2$.

From 2-OPSI($m_2$) to k-OPSI($m_k$)

Extending Protocol 5.2 to k-OPSI($m_k$) is possible but comes at the cost of reduced performance: In the results computation phase, the cloud peers compute shares $\langle B_{S, \cap} \rangle$ and $\langle B_{C, \cap} \rangle$ of the comparison bits for each cell of the simple and Cuckoo hash tables, respectively. We can compute shares of the matching elements or the dummy elements $d_1, d_2$ in the respective cells by using the shares of the comparison bits as inputs to a MUX circuit, i.e., for all $j = 0, ..., \beta - 1$ and $k = 0, ..., \text{max}_\beta - 1$:

$$
\langle H_{S, \cap}^{[j]} \rangle \leftarrow \text{GMW}_{\text{MUX}}\left(\langle B_{S, \cap}^{[j]} \rangle, \langle H_{S, 0}^{[j]} \rangle, d_1\right)
$$

$$
\langle H_{C, \cap}^{[j]} \rangle \leftarrow \text{GMW}_{\text{MUX}}\left(\langle B_{C, \cap}^{[j]} \rangle \oplus \ldots \oplus \langle B_{C, \cap}^{[\text{max}_\beta]} \rangle, \langle H_{C, 0}^{[j]} \rangle, d_2\right)
$$

This adds $l\beta(\text{max}_\beta + 1)$ non-linear gates to the circuit which approximately doubles the size of the original circuit. The results of these steps share the intersection $D_{1\cap} = D_1 \cap D_2$ of the two datasets in exactly the same hash table format as the outsourced datasets $D_1$ and $D_2$ and can be used in subsequent runs of the 2-OPSI($m_2$) protocol to compute $\langle (D_1 \cap D_2) \cap D_3 \cap \ldots \cap D_k \rangle$. To improve performance and minimize rounds, we arrange the sequential $k - 1$ invocations of 2-OPSI($m_2$) in a tree with $\lfloor k/2 \rfloor$ parallel invocations at level $i = 0, ..., \lceil \log_2 k \rceil$.

5.3.3 Outsourcing to a Single Peer

As presented in the previous section, outsourcing to two peers allows adapting the highly efficient 2-PSI protocols proposed in related work but requires data owners to choose two peers that they trust not to collude. It is thus interesting to examine whether we can eliminate this assumption, e.g., for scenarios with only a single data owner who outsources to a single cloud. Clearly, executing multi- or two-party protocols on a single peer breaks security as this peer could reconstruct all protected values, rendering all previous approaches inapplicable. Since we have also not identified a satisfiable approach to this setting in related work (cf. Section 5.2.2),
we propose two novel designs. We begin with a general k-
Protocol 5.3
Our novel combined k-OPSI
Output: Intersection Bloom filter $BF(\bigcup_{i=1}^{m} D_i)$
Preprocessing: for all $i = 1..m$
\[ U: \quad B_i \leftarrow CreateBF(l, D_i) \]
\[ B_i = ([b_{i,1}|...|b_{i,s_r}], [b_{i,s_r+1}|...|b_{i,1}]) \leftarrow Enc_{FHE}(B_i) \]
Setup:
\[ U \leftrightarrow C: \quad \text{Upload} \; \langle B_1, ..., B_m \rangle \]
Results computation:
\[ C: \quad [B_i] = [B_i] \oplus \cdots \oplus [B_m] \]
\[ C: \quad [D_i'] = [B_{i,1}] \oplus \cdots \oplus [B_{i,l/s}] \]
Postprocessing:
\[ U \leftrightarrow C: \quad \text{Download} \; \langle [D_i] \rangle \]
\[ U: \quad [D_i']' \leftarrow Dec_{FHE}([D_i']') \]
\[ U: \quad [D_i] \leftarrow |[D_i]/k| \]
\[ U: \quad B_i = (b_{i,1}, ..., b_{i,l}) \leftarrow Dec_{FHE}([B_i]) \]
\[ U: \quad D_i = \{ x_{i,j} \in D_i | \; B_i \text{ contains}(x_{i,j}) \} \]

Protocol 5.3: Our novel combined k-OPSI($\ell$) and k-OPSI-CA($\ell$) scheme based on FHE.

we propose two novel designs. We begin with a general k-OPSI($\ell$) scheme and then present a significant improvement for the special case of 2-OPSI-CA($\ell$). These two schemes were published in [ZPH17] as part of the BLOOM system for secure genomic matching that we present as our second case study in Section 5.5.

A novel k-OPSI($\ell$) protocol

Inspired by [MBD12,Ker12b], the core idea of our k-OPSI($\ell$) scheme is to represent each dataset as a Bloom filter and then compute the intersection (cardinality) under encryption on the single cloud peer using multiplication (and addition) over the Bloom filter bits. However, we need to employ an FHE scheme for the encrypted computations since we outsource to a single peer (as opposed to the multiparty setting assumed in [MBD12]) and realize the general case of $k \geq 2$ intersections (as opposed to $k = 2$ in [Ker12b]).

Protocol 5.3 provides the details of our k-OPSI($\ell$) scheme. In the preprocessing phase, the data owner encodes each dataset $D_i$ in a separate Bloom filter $B_i$. The data owner further chooses an FHE scheme, generates a key pair, and encrypts each Bloom filter bitwise.

Current FHE schemes support packing techniques which offer multiple plaintext slots within a single ciphertext and allow operating on the encrypted plaintexts in a SIMD manner [SV14,BGH13b,GHS12]. In the following, we denote a packed encryption of $s_F$ plaintexts by $[x_1|...|x_{s_F}]$ (not to be confused with $\lfloor \cdot \rfloor$, the notation of cardinality). We apply packing to the Bloom filter representations of each dataset, i.e., we encrypt $B_i$ in $[l/s_F]$ ciphertexts $[B_i] = ([b_{i,1}|...|b_{i,s_r}], ...,[b_{i,s_r+1}|...|b_{i,1}])$.

In the setup phase, the single data owner uploads and stores in the cloud all $m \cdot [l/s_F]$ ciphertexts $[B_1], ..., [B_m]$ that encrypt the respective datasets $D_1, ..., D_m$. 
In the results computation phase, the cloud multiplies the encrypted Bloom filters $[B_i]$ component-wise, i.e., $[B_1] \odot [B_2] \odot ... \odot [B_m] = [B_C]$ where $\odot$ denotes the encrypted multiplication operation of the FHE scheme. Note that we slightly abuse notation here, since this step actually requires $\lceil t/s_f \rceil$ parallel ciphertext multiplications, each of which carries out $s_f$ pairwise multiplications in an SIMD fashion. The resulting ciphertexts $[B_C] = ([b_{1,1}] | ... | [b_{1,s_f}] | ... | [b_{m,1}] | ... | [b_{m,s_f}])$ correspond to an encrypted Bloom filter in which exactly those bits are set that correspond to the Bloom filter representing the intersection $D_i = \cap_{k=0}^{m-1} D_k$. The cloud peer also sums up the $\lceil t/s_f \rceil$ ciphertexts in $[B_C]$ to compute an encrypted estimate of the intersection cardinality $||D_C|| = B_{C,1} \oplus ... \oplus B_{C,t/s_f}$.

In the postprocessing phase, the data owner finally downloads and decrypts the $\lceil t/s_f \rceil$ ciphertexts that encrypt the intersection Bloom filter $B_C$. The data owner then recovers the original elements from the Bloom filter by testing each element from a single set $D_i$ (preferably the smallest among all $m$ datasets) against $B_C$ (as explained before in Section 5.3.1). If only the cardinality of the set intersection should be computed, the data owner downloads and decrypts only the single ciphertext $||D_C||$ and scales down the result by $k$, as in our previous $k$-OPSI scheme. As before, we must bear in mind the possibility of false positives due to the use of Bloom filters as well as the slight underestimation of the true intersection cardinality due to an accumulation of hash collisions as quantified Section 5.3.4.

A novel 2-OPSI-CA protocol

We now concentrate on the special case of single-party outsourcing which reoccurs in different application scenarios. In this scenario, the data owner aims to delegate storage and processing of many datasets $D_1, ..., D_m$ to the cloud and, at an arbitrary time after the outsourcing, the data owner then needs to check all datasets individually for intersection cardinality with a single query dataset $D_Q$ which is typically much smaller than $D_1, ..., D_m$. A typical application scenario is found in the area of computational genomics as we demonstrate in detail in Section 5.5. In this example, a doctor securely outsources the sequenced genomes of many patients to the cloud (the datasets $D_1, ..., D_m$) and later queries them for different genetic variations.

The described scenario reduces to the $m$-times parallel invocation of 2-OPSI-CA which we can already handle with the previous $k$-OPSI-CA scheme. However, the special structure of the problem, i.e., $m$ parallel intersections of only two sets, allow us to significantly improve performance. As before, we represent all datasets and the query as Bloom filters, but instead of encrypting the query we now use a pre-image resistant keyed hash function to insert elements into Bloom filters. The query Bloom filter is then sent to the cloud without encryption since the keyed hashing already protects its contents. This enables a simpler intersection algorithm as well as a higher degree of aggregation before results are sent back to the client.

While the use of keyed hashing prevents the cloud from learning which set is queried, the cloud notices if the same query is posed twice. This presents a slight leakage of the data owners access patterns to her delegated datasets but achieves orders of magnitude performance increases. Importantly, all outsourced datasets are still fully protected through encryption. We now explain each step of our $m$-times parallel 2-OPSI-CA in detail. Note that we restrict ourselves to the computation of the
intersection cardinality – the scheme also allows computing the intersection itself but at higher costs for the data owner.

The preprocessing phase is very similar to the previous scheme with the only difference that the data owner uses keyed hashing with key $\kappa$ to insert elements into the Bloom filters $B$. This prevents any other party who is not in possession of the hashing key from determining which elements are contained in a given Bloom filter. In contrast to our first approach, we pack Bloom filter bits $b_{m,j}$ with all dataset Bloom filters $Q_1,...,Q_m$ and then uploaded without encryption, relying on key-hashing to protect its content. Since the cloud obtains $B_Q$ in the clear, intersecting $B_Q$ with all dataset Bloom filters $B_1,...,B_m$ is simply done by selecting those columns corresponding to set bits in the query, i.e., we select the encrypted column $[B^j]$ iff. the bit $b_{Q,j}$ is set. We thus retain a set of at most $|D_Q|-k$ encrypted columns $[B^{j_1(x_i)}],...,[B^{j_k(x_i)}]$ with $x_i \in D_Q$. To estimate the cardinality of the intersections of $D_Q$ with each $D_i$, the cloud sums the columns element-wise which is realized using encrypted additions that can be computed locally by the cloud due to the additive homomorphic property of the chosen PHE scheme. Aggregation

**Protocol 5.4** Our novel 2-OPSI-CA(1) based on PHE, optimized for the batched computation of the set intersection cardinality on two sets.
reduces the size of the results to a single encrypted column $[B^c]$ of only $\lceil m/s_P \rceil$ ciphertexts which significantly reduces the overheads for the data owner.

In the postprocessing phase, the data owner downloads and decrypts the $\lceil m/s_P \rceil$ ciphertexts in $[B^c]$. Each plaintext packs $s_P$ intersection cardinalities, i.e., the cardinality $|D_Q \cap D_i|$ can be simply read from slot $i \mod s_P$ of the $\lfloor i/s_P \rfloor$th plaintext in the decrypted results. As before, a slight underestimation of the true intersection cardinality must be accounted for as we quantify in Section 5.3.4.1.

### 5.3.4 Evaluation and Discussion

We qualitatively and quantitatively compare the presented OPSI schemes and analyze them with regard to the stated requirements for ideal outsourcing schemes (Section 5.2.1). Table 5.1 summarizes the results and includes the most promising existing outsourcing schemes for each scenario. Since we already analyzed related work in Section 5.2.2.2, we focus on the discussion of our schemes in the following.

#### 5.3.4.1 Support for OPSI and OPSI-CA

All our schemes except the highly optimized 2-OPSI-CA $(0)$ support the outsourcing of set intersections. The Bloom filter-based schemes, however, may produce false positives which are controlled through the false positive probability $p$. A lower $p$ requires more hash functions and longer Bloom filters, thus has a negative impact on performance which grows roughly in $O(-\log(p))$.

All our schemes support OPSI-CA, i.e., the computation of the cardinality of the set intersection without revealing the elements of the intersection to the data owners. In the single-party outsourcing scenario, computing only the cardinality even significantly reduces the communication and processing overheads for the data owner. The k-OPSI-CA $(0)$ scheme computes the exact cardinality while all other schemes compute only an estimate as explained in the following.

For our k-OPSI-CA $(0)$, k-OPSI-CA $(1)$, and 2-OPSI-CA $(1)$ schemes, we rely on a Bloom filter representation of the intersected sets and count the set bits to compute the cardinality of the intersection. This approach, however, does not compute the

### Table 5.1 Summary and comparison of prior proposals (top) and our own OPSI schemes (bottom) with respect to our requirements and design goals.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>k-OPSI $(0)$ [BA16]</td>
<td>Exact</td>
<td>Exact</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>2-OPSI $(2)$ [Ker12b]</td>
<td>FP $p$</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>k-OPSI $(1)$ [ATD15]</td>
<td>Exact</td>
<td>(✓)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>k-OPSI $(m)$</td>
<td>FP $p$</td>
<td>Estimate</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>2-OPSI-CA $(1)$ [ZHBW17]</td>
<td>Exact</td>
<td>Exact</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>k-OPSI-CA $(1)$</td>
<td>FP $p$</td>
<td>Estimate</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>2-OPSI-CA $(1)$ [ZPH 17]</td>
<td>✓</td>
<td>Estimate</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
5. Outsourced Private Set Intersection

Exact intersection cardinality since hash collisions may occur when inserting elements into the Bloom filters, i.e., $H_i(x) = H_j(x')$ for some $1 \leq i, j \leq k$ and $x \neq x' \in U$ which results in only a single bit set to one instead of two. Thus, our approach underestimates the true cardinality by exactly one when more than $k$ hash collision occur, by two when more than $2k$ collisions occur, and so forth. We hence need to analyze the expected number of hash collisions in the intersection of the sets to quantify how badly we underestimate the true cardinality of the intersection. When inserting $n$ elements using $k$ into a Bloom filter with $l$ slots, the expected number of collisions according to the Birthday problem is:

$$E_{\text{collision}}(n, k, l) = nk - l \left( 1 - \frac{1}{l} \right)^nk$$

(5.1)

Since our OPSI-CA schemes compute the set intersection by counting set bits in the intersection Bloom filter, dividing the result by the number of hash functions $k$, and rounding up, the true cardinality is underestimated by at most

$$Err_{\text{OPSI-CA}} = \left\lfloor \frac{E_{\text{collision}}(n, k, l)}{k} \right\rfloor$$

(5.2)

Figure 5.2 plots the true cardinality (green) versus the estimate of our OPSI-CA schemes. The x and y axes thus denote the true and estimated cardinality relative to the size of the original sets as the relative sizes (and errors) are the same for all set sizes $n$ and choices of the Bloom filter’s false positive probability $p$ when choosing the number of hash functions $k$ and the length of the Bloom filter $l$ optimally as detailed in Section 5.3.1. We observe that the relative error increases as the relative size of the intersection increases which is as expected as a larger intersection increases the probability of collisions superlinearly, e.g., if the intersection contains only 25% of the elements of the original sets, we underestimate the cardinality by only 8.22%, while the error increases to 22.05% when the intersections contains 75% of the elements of the original sets. When we expect intersections to be small our OPSI-CA approach is thus reasonably accurate, e.g., in our second case study (Section 5.5) it has no effect, at all. When the relative intersection size and relative errors are expected to be larger, it must be decided depending on the use case whether the inaccuracy can be tolerated – to increase accuracy in these cases, we
Finally, we emphasize that our approach always computes correctly the Boolean decision whether the intersection is empty or non-empty despite the quantified errors. For some application scenarios of OPSI-CA this is already sufficient, e.g., for the location privacy mechanism proposed in our first case study (Section 5.4) or for special cases of the genetic testing application in our second case study (Section 5.5).

5.3.4.2 Scalability

We provide a brief and preliminary comparison of the runtime and communication overheads of our proposed outsourcing schemes. A more detailed evaluation of the adapted 2-OPSI-CA($^m_2$) and our novel k-OPSI-CA($^1_1$) and 2-OPSI-CA($^1_1$) schemes will be provided as part of our two case studies (Sections 5.4 and 5.5).

**Experimental setup.** For the performance comparison, we implemented all proposed OPSI schemes and measured the runtime and communication overheads for i) the data owner for outsourcing their datasets and ii) the cloud peers to compute the intersection on sets of different cardinalities. For a direct comparison, we limit the number of intersected datasets to $k = 2$ (overheads increase linearly in $k$ for all schemes except for 2-OPSI-CA($^1_1$) which only supports $k = 2$) and compute only the cardinality of the set intersection. For the multi-party outsourcing scheme k-OPSI-CA($^m_o$) to remain competitive, we choose a small number $o = 5$ and $o = 7$ of cloud peers. All set elements have a length of $l = 32$ bit and Bloom filters are configured for a false positive probability of $p = 2^{-14}$. To allow packing strategies to fully play out, we compute batches of pairwise intersections and provide the mean runtime and communication overhead for a single intersection operation from these batches. Data owners and cloud peers are run on different desktop machines with the same soft- and hardware (Ubuntu 14.04, Intel i7-4770 at 3.10 GHz, 16 GB RAM). For simplicity, all parties are connected over a Gigabit LAN. We provide the mean and standard deviation over 30 runs.

**Overheads for data owners.** The runtime and communication overheads for data owners for preprocessing, setup, and postprocessing of all schemes are plotted and
5. Outsourced Private Set Intersection

Figure 5.4 Comparison of the runtime (left) and communication (right) overheads for the cloud for the outsourced computation of the intersection of two datasets of varying sizes (x-axis).

compared in Figure 5.3. We observe that runtime and communication overheads of all schemes scale linearly in the set size (note the logarithmic scale of the x-axis). 2-OPSICA\((m_2)\) is most efficient, its overheads being almost negligible. Despite its use of public key cryptography, our novel 2-OPSICA\((1)\) is the second most efficient scheme due to its efficient packing strategy. For outsourcing to many peers, we notice that overheads increase linearly in the number of peers to which computations are outsourced (i.e., k-OPSICA\((m)\) versus k-OPSICA\((m_7)\)) and are at least twice as high as for 2-OPSICA\((1)\). Finally, k-OPSICA\((1)\) imposes the highest runtime and communication overheads on the data owners which is due to the use of more expensive FHE.

Overheads for cloud peers. Figure 5.4 plots runtime and communication of the cloud peers for executing the results computation phase. Our 2-OPSICA\((1)\) is the most efficient scheme by orders of magnitude followed by our more general k-OPSICA\((1)\) variant. Since both schemes outsource to a single cloud peer, their result computation phase requires no communication which makes them immune to network latency in contrast to the other schemes in the multi- or two-party setting. Perhaps surprisingly, k-OPSICA\((m_5)\) is slightly faster than 2-OPSICA\((m_2)\) which is mainly due to the different underlying set representation and resulting intersection protocol (i.e., Bloom filters versus hash tables). Again we observe significantly increased overheads when outsourcing to more peers in k-OPSICA\((m)\).

5.3.4.3 Security

We show that our outsourcing schemes are secure in the semi-honest model and discuss how many colluding cloud peers (or data owners) each scheme can tolerate.

k-OPSICA\((m)\) (cf. Protocol 5.1). The only interaction between data owners in the preprocessing phase is in announcing the chosen cloud peer which has no security implications in the semi-honest model. However, we emphasize that this message should be signed and verified to prevent a man-in-the-middle attacker on the communication path from substituting \(U_i\)'s choice \(C_i\) with a cloud peer under the attacker’s control.
In the setup phase, the data owners distribute secret shares of their inputs to the cloud peers using a $t$-out-of-$o$ linear secret sharing scheme which tolerates collusions of $\tau < t$ cloud peers – any subset of $\tau \geq t$ colluding cloud peers can completely recover the input datasets.

The first two steps of the results computation phase involve multiplication and addition over secret shares - addition is only a local operation and multiplications protocol have been proven secure in [ALR11]. However, the multiplication protocol imposes the limit of $2t-1 \leq o$ as explained in Section 2.3.3.1. The third step involves the DivideAndCeil protocol which is composed of building blocks proven secure in [CS10]. All steps operate completely over secret shares such that any subset of $\tau < t$ colluding cloud peers learns nothing.

In the postprocessing phase, cloud peers return the secret-shared output to the data owners who can recombine the result locally. If only the set intersection cardinality should be computed, the cloud peers simply do not send back $\langle B \cap C \rangle$, which prevents the data owners from learning the elements of the intersection. We conclude that our combined k-OPSI$m$ and k-OPSI-CA$m$ scheme is secure in the semi-honest model against $\tau < o/2$ colluding cloud peers.

**k-OPSI$^m$ (cf. Protocol 5.2).** In the preprocessing phase, data owners locally compute the hash table representation of their sets and use 2-out-of-2 Boolean secret sharing to securely distribute these hash tables to the two cloud peers in the subsequent setup phase. In the standard two-party setting where data owners execute the protocol, non-collusion is a standard assumption since a data owner would also harm herself by colluding with the other party. In our outsourcing settings, cloud peers could collude to attack the data owners’ privacy, thus data owners must agree on cloud peers that they trust not to collude. As frequently argued in the related literature [KMRS14,ATD15], it is reasonable to assume that established commercial cloud services will not collude in order to protect their reputation and business as computation and storage providers. To further ensure non-collusion, cloud peers could be placed in different legislative administrations which could even prevent forced access from governments.

In the results computation phase, we basically execute subparts of the protocol proven secure in [PSSZ15] which we only modified by circuit components that count the cardinality of the intersection and return the result of the intersection in the hash table representation for subsequent use. The security of these steps follows directly from the security of the underlying GC or GMW protocol for securely evaluating the Boolean circuits (cf. Section 2.3.2).

In the postprocessing phase, cloud peers return to the data owners the shares of the intersection hash table (either in the simple or cuckoo hashing format) which allows the data owners to recover only the elements in the intersection as desired. If only the set intersection cardinality should be computed, the cloud peers do not send back the shared hash tables, which effectively prevents the data owners from learning the elements of the intersection. We conclude that our k-OPSI$^m$ outsourcing scheme based on the 2-PSI protocol by Pinkas et al. [PSSZ15] is secure in the semi-honest model when cloud peers do not collude (i.e., $\tau < 2$ colluding peers).

**k-OPSI$^1$ (cf. Protocol 5.3).** Preprocessing, setup, and postprocessing include only the single data owner (except for the up- and download of encryptions). These
phases are trivially secure in the single-party setting since everything that the single data owner learns from the protocol run can be learned from her inputs anyways. We thus only have to consider security of the results computation phase. Since the data owner only uploads encryptions of her data and the cloud operates exclusively on these ciphertexts without interaction with the data owner, the security of the results computation phase follows directly from the security of the chosen underlying FHE scheme. Notably, since we just up- and download ciphertexts to a single party, our scheme is trivially secure against collusion.

2-OPSI-CA \(^{(1)}\) (cf. Protocol 5.4). Preprocessing, setup, and postprocessing involve only the single data owner (except for the up- and download of encryptions) and are trivially secure as argued before. Different to the previous scheme, the query \(D_Q\) is not encrypted. Instead, we use a pre-image resistant keyed hash function to map set elements into the Bloom filters and only the data owner knows the hashing key \(\kappa\). The use of a secret hashing key prevents the cloud from mounting a brute-force attack to learn which elements are queried in \(D_Q\). Note that we use the key only to salt the hash and do not require, e.g., robustness against length-extension attacks as provided by keyed hash functions for message authentication such as HMAC [KBC97]. However, we require the keyed hashing to be deterministic which breaks semantic security. In consequence, the cloud is able to distinguish whether queries \(D_{Q_1}\) and \(D_{Q_2}\) are different, the same, or how much they overlap. The following results computation phase is then computed completely under semantically secure encryption and no further information is leaked. We thus conclude that our 2-OPSI-CA \(^{(1)}\) scheme fully secures outsourced data and computed results, is trivially secure against collusion, and only slightly leaks access patterns when multiply queries are posed. We argue that the latter limitation is tolerable in scenarios where high performance is paramount.

### 5.3.4.4 Repeated Independent Execution

In all our outsourcing schemes, the data owners encode and upload the protected datasets exactly once and cloud peers can at any time afterwards compute an unlimited number of arbitrary intersections on them without any need to involve the data owners. In the \(k\)-OPSI\(^{\text{\(m\)}}\) scheme, this is achieved by outsourcing datasets in both the simple and cuckoo hash table formats.

### 5.3.4.5 Composability

All our schemes achieve only limited composability since they use special set representations, i.e., Bloom filters and hash tables. These data structures are non-trivial to obtain from or transform back to the original set elements and are thus not universally usable in subsequent secure computations. However, our schemes are directly composable with any secure protocol that operates over the same set representation. Composability for our OPSI-CA schemes is less limited as we obtain shares or encryptions of the set cardinality which can be subsequently used almost as is. For the Bloom filter-based schemes, however, we need to compute the division by the number of hash functions \(k\) and the rounding securely in the cloud when the cardinality should not be returned to the data owners but reused in another secure
5.3. Outsourced Private Set Intersection

computation. This can be implemented at moderate additional overheads using the secure non-integer building blocks due to Catrina and Saxena [CS10].

The only scheme that achieves composability without any restrictions is the k-OPSI(\(m\)) protocol due to Blanton and Aguiar [BA16] which uses the original set elements as inputs and outputs. Full composability, however, comes at significant performance overheads: Blanton and Aguiar’s approach [BA16] requires approximately 22.5s for the intersection of two sets of size \(2^{11}\) elements in a small network of \(m = 3\) cloud peers. In comparison, our k-OPSI-CA(\(m\)) approach requires only 2.3s for the same task. It must be decided depending on the concrete use cases whether full composability is required and worth the one order of magnitude decrease in performance of Blanton’s compared to our scheme.

5.3.4.6 Flexible data handling

In all our schemes, data owners may outsource datasets at arbitrary times prior to computing intersections. In our 2-OPSI-CA(\(1\)) scheme, data owners should upload batches of \(s_P\) datasets, which minimizes overheads with respect to the employed packing strategy. To delete an outsourced dataset, data owners simply instruct the cloud peers to delete the corresponding shares or ciphertexts. In all schemes, an outsourced dataset can also be modified without leaking access pattern simply by replacing it.

Data owners can also update only single Bloom filter or hash table positions, which leaks corresponding access patterns but minimizes overheads. In k-OPSI(\(m\)) and k-OPSI(\(m\)), data owners can update each position of the Bloom filter individually by simply re-encrypting and uploading the respective bits. In our k-OPSI(\(1\)) scheme, we pack \(s_F\) bits of a single dataset into a single ciphertext (row-wise), which allows updating multiple Bloom filter bits in one operation. In our 2-OPSI(\(1\)) scheme, we pack \(s_F\) bits over different datasets (column-wise) into a single ciphertext. While this packing strategy is less efficient for modifying single datasets, it allows the data owner to operate on batches of \(s_P\) outsourced datasets in an SIMD fashion.

We remark that additional care must be taken when deleting elements from Bloom filters: If a hash collision occurred when adding two elements, deleting the \(k\) positions corresponding to one item (i.e., setting those positions to zero) then also deletes the colliding position for the other item. Data owners must thus store elements which caused hash collisions and keep the corresponding Bloom filter positions untouched until the colliding element is deleted as well.

5.3.5 Summary and Future Work

We have presented four OPSI and OPSI-CA schemes that compare favorably to prior works. In our designs, we prioritize efficiency over generality which leads to much better scalability at the cost of limited composability and exactness compared to, e.g., the proposal of [BA16]. Ours are the first among all analyzed existing schemes to achieve outsourcing from a single data owner to a single party without any non-collusion assumptions. In addition to these notable contributions, we identify the following directions as worthy and important future work:
k-OPSI(\(m\)) with collusion resistance: The only schemes [Ker12b, ATD15] that allow multiple data owners to outsource to a single cloud peer require that the cloud does not collude with any data holder (noted by \(\tau < 2\) in Table 5.1). Indeed, we can realize k-OPSI(\(m\)) from our k-OPSI(1) scheme in the same security model. From a security perspective, it is an interesting challenge to design k-OPSI(\(m\)) with stronger robustness against such collusions.

Efficiency vs. correctness: We achieve efficient protocols partly due to the use of probabilistic data structures, i.e., Bloom filters and hash tables. False positives in Bloom filters can lead to false positives in the computed set intersection and the approaches based on hash tables may fail when too many collisions occur. We have also analyzed how the accumulation of hash collisions leads to an underestimate of the true cardinality. In our approaches, we can thus trade-off efficiency against accuracy through the tunable false positive probability \(p\).

An interesting line of research is to rigorously investigate the trade-off between efficiency and accuracy of PSI and PSI-CA as well as to demonstrate use cases for such secure approximations of set intersection (cardinality).

Efficiency vs. composability: We trade off composability against efficiency through the use of special representations of sets, i.e., Bloom filters and hash tables. However, composability is an important property as PSI and OPSI schemes are mostly used as a building block in more complex secure computations. Future work should thus investigate how to better preserve composability of OPSI schemes while maintaining high efficiency.

Comparable performance benchmarking: As similarly noted by the authors of [PSZ14], we observe that comparing PSI and OPSI schemes is very difficult due to i) the large number of existing proposals, ii) missing implementations for many proposals, iii) varying security assumptions and scenario requirements, as well iv) different design goals, optimization targets, and evaluation setups. Given the tremendous progress made in PSI, we consider dependable, comparable, and comprehensive cross-evaluations an important step to further advance this research field.

5.4 Case Study: Crowd-Sensing

In this section, we present our first case-study in order to validate the applicability, scalability, and adaptability of our OPSI schemes. As use case, we motivate the need for effective protection of location privacy in crowd-sensing (Section 5.4.1) and point out inherent special requirements of crowd-sensing that hitherto remain unaddressed by related work (Section 5.4.2). To address these novel requirements, we present the TraceMixer system [ZHBW17], a novel location privacy protection mechanism (LPPM) that implements a mix-zone-based anonymization approach on top of our 2-OPSI(\(m\)) scheme. Our OPSI protocol is the core building block that allows TraceMixer to operate over the outsourced location data of dynamically joining and leaving mobile participants securely such that sensitive location traces are never shared in clear with anyone (Section 5.4.3). The implementation, evaluation, and deployment of our prototype on standard of the shelf smart phones in
a real-world sensing campaign for the privacy-preserving creation of high precision elevation profiles using barometric altitude show the feasibility of TraceMixer in terms of performance, location privacy, and data utility (Section 5.4.4). In conclusion, with TraceMixer we not only demonstrate the real-world applicability of our developed 2-OPSI-CA (\(m^2\)) scheme but present an independent contribution to the important challenge of providing privacy protection to crowd-sensing and other emerging participatory computing paradigms (Section 5.4.5).

5.4.1 Motivation

The proliferation of mobile, location-aware sensing devices has enabled and inspired crowd-sensing, a new participatory computing paradigm. The core idea of crowd-sensing is to tap into the sensing capabilities of ordinary smart phones for distributed large scale data collection. Thereby, crowd-sensing promises higher coverage and availability at lesser costs than traditional data collection methods. Successful real-world crowd-sensing campaigns range from collaborative map creation [OSM17] to monitoring cellular networks [Ope17] or road conditions [MPR08]. Unsurprisingly, crowd-sensing is receiving increasing attention also from commercial players: E.g., the cellular network operator Telefonica sells insights to retailers based on its customers’ location data [BBC12], while Strava, a site for tracking sport experiences, promises to support and improve city planning based on its users’ activity traces [Str14].

Common to most crowd-sensing campaigns is the focus on the spatial context of contributed data, e.g., in all previous examples data is annotated with a location or the location information itself is the main item of interest. However, past and recent studies [BD03,Inc10,AA14] show that users have serious privacy concerns over disclosing location data and numerous attacks on location data as well as serious real-world incidents [Kru07,GKdPC14,Sch10] show that users’ location privacy concerns are well justified. Indeed, location privacy concerns have been identified as a main obstruction to crowd-sensing [CRKH11,GMS+13,ZGMW14]. Effective LPPM are hence required to resolve privacy concerns among participants in order to realize the full potential of the crowd-sensing paradigm.

The majority of existing LPPMs are tailored to traditional location-based services (LBSs). In an LBS, users annotate queries with their locations and receive answers personalized to their local scope. Despite a shared focus on location information, crowd-sensing has fundamentally different characteristics than traditional LBS and hence poses different requirements to LPPMs. First, LBSs usually involve the disclosure of only single locations while crowd-sensing campaigns typically collect whole traces. LPPMs for crowd-sensing must thus provide trajectory privacy instead of the simpler and weaker notion of sporadic location privacy. Second, LBSs value the timeliness of answers over their spatial accuracy. In contrast, many crowd-sensing campaigns such as map creation [OSM17] or city planning [Str14] have rather long-term goals and favor spatial over temporal accuracy. Unfortunately, this mismatch of privacy notions, and utility goals renders LPPMs developed for LBS unsuitable for many crowd-sensing scenarios.

In addition to the mismatching privacy and utility goals, we observe that existing LPPMs typically require a Trusted Third Party (TTP) that acts as an anonymization
proxy and learns all data in clear. As we have argued before (Chapter 1), such central entities are vulnerable to attacks, database leaks, or access by governments. This is even aggravated by the long-term data collection and storage typical in crowd-sensing. We thus argue that LPPMs must renounce centralized design patterns and should ideally never learn participants’ data in clear.

This case study starts from the observation that the mismatching privacy and utility goals of existing LPPMs require the design of novel mechanisms tailored to the special requirements of crowd-sensing. While SMC is a promising approach to resolve the need for a TTP in such LPPMs, we expect significant challenges when applying SMC within crowd-sensing that motivate the use of outsourcing schemes: Participating users in crowd-sensing campaigns are typically mobile, their devices have constrained processing and storage resources, communicate over constrained networks, and are dynamically joining and leaving. In the following section, we frame the problem of location privacy in crowd-sensing in detail.

5.4.2 Problem Statement

Figure 5.5 illustrates our abstract model of crowd-sensing campaigns in three phases, i) data collection, ii) privatization, and iii) publication. Data collection is initiated by the campaign administrator by instructing the participants with a sensing task. The participants, referred to as users $U_i \in U$, move around in an area divided into discrete locations. Each user is equipped with a location-aware device and continuously collects data reports along her movement trajectory. A report is defined by its location and the sensed event. Events can be anything from single sensor readings such as noise levels [MSNS09] to arbitrarily sophisticated (sensed) phenomena such as road conditions [MPR08, GPA10]. Events may be empty, e.g., when location information is the main target of data collection [BBC12]. Finally, the users upload reports to the LPPM continuously or in batches. In the second phase, privatization, the LPPM is responsible for applying adequate privacy protection before releasing any data. We emphasize that our focus is on privacy protection regarding the location information in the uploaded reports which is found in the majority of crowd-sensing campaigns. We deliberately do not address potential privacy risks concerning the reported events themselves since they are specific to each category of sensed data, e.g., audio data requires different protection than temperature readings [CRKH11]. In the final publication phase, the LPPM releases the privatized data to the interested
parties. Since attackers may disguise as benign applications, the LPPM has to trade off data utility against privacy.

We survey different architectures in the literature as well as real-world deployments to distill the unique requirements of crowd-sensing. These requirements distinguish crowd-sensing from traditional LBS and motivate the need for novel LPPMs.

**Privacy notion:** In the majority of the surveyed crowd-sensing systems and deployments, the users upload successive reports which form mobility traces, e.g., in road condition monitoring [MPR08], map creation [OSM17, GPA+10], or city planning [Str14]. Disclosing whole traces involves significantly higher privacy risks than sporadically disclosing only single locations as typical for traditional LBS. Users exhibit unique mobility patterns which can be exploited to re-identify anonymously contributed traces, predict users’ future whereabouts, or infer sensitive information such as users’ points-of-interest [GH05, Kru07, GKdPC14]. Hence, LPPMs for crowd-sensing must provide trajectory privacy instead of the weaker notion of sporadic location privacy assumed in the context of traditional LBS.

**Utility goals:** Due to the voluntary, unreliable nature of crowd-sensing, many campaigns focus on data collection about stable phenomena such as long-term mobility patterns [Str14, BBC12], locations of roads and buildings [OSM17], or physical road conditions [GPA+10]. At the same time, these examples make evident the need for high spatial accuracy of contributed data, e.g., mapping roads and buildings requires spatial accuracy in the order of meters or even centimeters [OSM17]. In contrast, the query-response model of traditional LBS usually requires timely operation. This has led to the development of LPPMs that minimize temporal delays and resort to spatial distortion to provide privacy. Existing LPPMs thus optimize for utility goals opposite to those in the typical long-term crowd-sensing campaigns considered in this work. We conclude that adequate LPPMs for crowd-sensing should optimize spatial over temporal accuracy.

**Trust model:** Finally, we observe that most existing LPPMs are operated as centralized anonymization proxies which learn the locations of all users in clear. However, such central entities are vulnerable to attacks, database leaks, and forced access by governments [Pri17] — the long-term collection and storage of huge data sets makes crowd-sensing deployments particularly attractive targets. In consequence, users are hesitant to participate due to privacy concerns [CRKH11, GMS+13]. We thus argue that LPPMs in general and especially for crowd-sensing should renounce centralized design patterns and ideally never learn users’ locations in clear to minimize attack vectors.

### 5.4.2.1 Related Work

We analyze whether related work fulfills the stated requirements and compare it qualitatively to our own approach, TraceMixer. Due to the huge amount of literature on sporadic location privacy, we concentrate in this case study on those approaches that fulfill our first requirement, trajectory privacy.
5. Outsourced Private Set Intersection

Mix zones. The concept of mix zones [BS03] is to introduce quiet zones in which users do not report locations. In analogy to anonymous communication networks, users must stay in mix zones for a certain time so that the entry and exit events (i.e., users entering and leaving the mix zone) become unlinkable to outside observers. Several works improve upon the design and placement of mix zones, e.g., [FSH09, LZP+12, GMS+13]. An interesting property of this approach is that spatial data is reported accurately outside of mix zones which fulfills our utility requirement.

TraceMixer is also based on the idea of mixing users but introduces two key differences compared to the traditional concept of mix zones: Prior proposals place mix zones apriori, hence cannot guarantee that users will actually mix. In contrast, our approach checks aposteriori and obliviously (using PSI) whether a certain number of users have mixed and releases corresponding data only then.

k-anonymity. Different approaches adapt k-anonymity [Swe02] to location data by aggregating k traces such that they become indistinguishable to an attacker [CM07, PMX09, NAS08, ABN10]. These approaches introduce significant spatial distortion which violates our utility requirement. Further, they require either a TTP which violates our trust model or require direct interaction between users which is difficult to achieve in the crowd-sensing scenarios targeted in this work, e.g., we observe only rare encounters between users in the (popular) datasets used in our evaluation.

In TraceMixer, instead of spatially distorting traces, we obliviously check whether traces intersect (using PSI) and release only aggregates of k intersecting traces. We thus adopt the notion of k-anonymity as a measure of privacy for the released data but provide a secure protocol to establish it (based on STC).

Differential privacy. A recent line of research on trajectory privacy [AHS12, ABCP13, CPS14] derives from the differential privacy framework [DMNS06]. Basically, traces are privatized by adding carefully calibrated noise. Approaches based on differential privacy generally provide stronger privacy than approaches based on k-anonymity and can also be implemented in a user-centric fashion without a TTP. Due to the inevitable spatial distortion introduced by the addition of differentially private noise our utility requirements are, however, not met. For example, in [ABCP13] the added noise grows linear in the number of disclosed locations and greatly degrades utility even of short traces.

 Uncertainty. Different approaches use the uncertainty of a tracking adversary as privacy metric and optimization goal. Uncertainty is achieved, e.g., through path confusion [HG05] or path cloaking [HGXA07]. Both approaches require TTPs and significantly decrease data utility due to spatial distortion and data loss, respectively. In [TST+14], Bayesian Stackelberg games are used to determine an optimal trade-off between utility and privacy. Being completely user-centric, the latter framework fits our trust model but is limited to spatial perturbation as privatization mechanism which contradicts our utility goals.

The idea of measuring location privacy by the success of a tracking adversary as initially proposed in [HG05, HGXA07] has been widely adopted in related work and we also measure the privacy achieved by TraceMixer accordingly. However, TraceMixer uses different means of creating uncertainty than [HG05, HGXA07, TST+14] in order to preserve the spatial accuracy of contributed data.
5.4. Case Study: Crowd-Sensing

**Dummy trajectories.** In this class of approaches, either a TTP or the users introduce a large set of dummy trajectories that are similar to the real trajectories in order to hide the real trajectories within the mass of dummy trajectories [CG09]. This approach is unsuitable to crowd-sensing since the large amount of fake dummy reports cannot be distinguished from the real reports with the consequence that the large amount of fake measurements is likely to destroy the utility of the sensed data.

**Other approaches.** As summarized in [CRKH11], many proposals consider anonymous reporting as the main solution to protecting location privacy in crowd-sensing. However, anonymizing identities alone does not provide sufficient privacy since location traces can often be re-identified and reveal sensitive information [GH05, Kr07, GKdPC14]. Like TraceMixer, the approach in [TM08] is motivated by the goal to achieve high spatial accuracy. Their optimization algorithm requires a TTP with global knowledge of all traces and considers the inference of single locations instead of tracking attacks, i.e., assumes a different privacy notion.

We conclude that none of the related works proposes an LPPM that fully addresses the special requirements of the crowd-sensing scenarios targeted in this case study.

5.4.3 TraceMixer Design

We now explain the design of TraceMixer and how it addresses the special requirements of crowd-sensing. We start from a high-level system overview, then explain each component and their interaction in detail.

On the highest level, TraceMixer follows traditional anonymization approaches: Data records are collected from a number of sources, anonymized through an adequate mechanism, and finally released to the public in privatized form. Departing from all previous approaches, TraceMixer anonymizes traces obliviously and without spatial distortion. Our core idea that facilitates these key differences is to employ PSI to obliviously find sets of intersecting traces that can be safely released without anyone learning a participant’s trace in clear. Here, the intuition is that intersections of traces form natural mix zones which prevent attackers from tracking users if enough such mix zones are present in a released aggregate. PSI ensures that individual traces are never revealed in clear and that even the LPPM remains oblivious to them. Since our application scenario involves mobile users with constrained devices that dynamically join and leave, we opt for our OPSI schemes to overcome these challenges that render traditional PSI inapplicable.

Figure 5.6 provides an overview of the complete TraceMixer system. At the core is the oblivious anonymization protocol which is divided into two parts: First, OBLIVIOUSAGGREGATION determines sufficiently large sets of intersecting traces using our 2-OPSI\textsuperscript{(m)} scheme. Second, once such a set is found, SHUFFLERELEASE is used to shuffle, decrypt, and release the traces. The task of the user side component is the discretization of traces (Step 1, Section 5.4.3.1) and the preparation of traces for the two parts of the anonymization steps: Hashing and Sharing (Step 2.1) denotes the preprocessing and setup phase of the OPSI protocol that is the core of OBLIVIOUSAGGREGATION (Step 2.2), which is run collaboratively between two untrusted cloud peers \(C_1\) and \(C_2\). Trace Encryption (Step 3.1) prepares data for SHUFFLERELEASE (Step 3.2). For clarity, we explain Step 2 and Step 3 as a whole
5. Outsourced Private Set Intersection

Figure 5.6 Overview of TraceMixer: On the user side, traces are collected and preprocessed for the anonymization mechanism. The oblivious anonymization protocol is executed by two cloud peers, \( C_1 \) and \( C_2 \), which determine aggregates of intersecting traces that can be safely released. Finally, the achieved level of privacy is measured by the success of a tracking attacker with background knowledge about users’ mobility patterns.

in Sections 5.4.3.2 and 5.4.3.3, respectively, although the individual substeps are executed at different points in time and by different parties. Finally, in Step 4 (Section 5.4.3.4), we quantify how much privacy this anonymization approach actually achieves. To this end, we propose a state of the art tracking adversary whose success probability serves as our privacy metric.

5.4.3.1 Discretization

The goal of the discretization step is to map traces from continuous Global Positioning System (GPS) coordinates into a discrete location space. This serves two purposes: First, raw GPS data is error-prone and for many use cases overly precise. Discretization smooths out errors and allows adjusting to the required granularity. Second, we use OPSI to find intersections of traces which requires discrete locations. At first sight, the use of OPSI seems to limit our approach. However, discretization introduces artificial intersections, e.g., between two traces that run close but without intersections, which is desirable as it increases privacy.

To discretize continuous GPS samples, users locally map GPS coordinates to nearby reference points on OpenStreetMap (OSM). To put an upper limit on the discretization error, we only map to OSM nodes that are within a maximum distance of 2m and discard other locations. All further steps are now carried out over the discretized traces which are represented as lists of OSM node IDs.

5.4.3.2 Oblivious Aggregation

Protocol 5.5 shows the details of ObliviousAggregation, our secure anonymization protocol which finds aggregates of at least \( k \) intersecting traces that satisfy the \( k \)-anonymity privacy metric and can be safely released. We start with an empty set of aggregates \( \mathcal{A} = \{ \} \). For each outsourced trace \( T_i \in \mathcal{T} \), we greedily search for an
5.4. Case Study: Crowd-Sensing

Input: Shared traces \( \langle T \rangle = \{ \langle T_1 \rangle, \ldots, \langle T_n \rangle \} \), \( A = \{ \} \), \( k \in \mathbb{N}^+ \)

Output: Aggregates \( A_1, \ldots, A_m \) with \( A_l \subseteq \{ 1, \ldots, |T| \} \) and \( |A_l| \geq k \)

\[ C_1 \iff C_2 : \text{for all } \langle T_i \rangle \in \langle T \rangle \text{ for all } A_l \in A \text{ for all } j \in A_l \]
\[ \langle |T_i \cap T_j| \rangle \leftarrow \text{OPSI-CA}(\langle T_i \rangle, \langle T_j \rangle) \]
\[ \langle |T_i \cap T_j| > 0 \rangle \leftarrow \text{GC} \]
\[ |T_i \cap T_j| > 0 \leftarrow \text{Recombine}(\langle |T_i \cap T_j| > 0 \rangle) \]
\[ A_l.\text{add}(i) \]
\[ \text{if } |A_l| \geq k \]
\[ A_l.\text{pop}(A_l) \]
\[ \text{ShuffleRelease}(A_l) \]
\[ \text{goto start} \]

\[ A.\text{add}(\{ i \}) \]

Protocol 5.5 The ObliviousAggregation protocol: Cloud peers repeatedly engage in 2-OPSI\(^2\) to obliviously find aggregates of at least \( k \) intersecting traces that can be released.

aggregate \( A_l \in A \) that intersects \( T_i \). An aggregate \( A_l \) intersects a trace \( T_i \) if at least one trace \( T_j \in T_l \) \( j \in A_l \) intersects \( T_i \). To preserve the privacy of the users, the cloud peers operate over the shares \( \langle T \rangle \) of the traces and use our 2-OPSI-CA\(^2\) scheme to securely test two shared traces \( \langle T_i \rangle \) and \( \langle T_j \rangle \) for intersection. We emphasize that repeatability of the used OPSI-CA scheme is a vital property (cf. Section 5.2.1) for this approach since we must not depend on users to re-encode datasets for each new intersection (as required in some prior works) as they may be offline, connect over constrained devices and networks, or be generally unavailable for interaction.

Since our OPSI-CA scheme returns a share of the intersection cardinality, we test in a subsequent GC whether the cardinality is greater than zero in order to minimize the information contained in the recombined result (made possible by the composability of our OPSI-CA scheme). This way, cloud peers only learn whether two traces intersect but nothing else – not even in how many locations they intersect. If an intersecting aggregate \( A_l \) is found, we add the shared trace \( \langle T_i \rangle \) to it (i.e., its internal identifier \( i \)). As soon as the privacy criterion \( |A_l| \geq k \) is fulfilled, we remove \( A_l \) from the set of aggregates \( A \) and hand it to the second part of our anonymization mechanism which shuffles traces, then decrypts and releases the aggregate, after which the original shares of the traces in \( A_l \) are deleted by the cloud peers. If no aggregate that intersects with the received trace \( T_i \) is found, we create a new aggregate that only contains the shared trace \( \langle T_i \rangle \).

5.4.3.3 Shuffle and Release

In the previous step, we have determined aggregates \( A_l = \{ i_1, \ldots, i_k \} \) of intersecting traces \( T_{i_1}, \ldots, T_{i_k} \) without ever looking at the traces in clear. These aggregates \( A_l \) now need to be released safely. However, cloud peers only hold shares of the original traces and recombining them directly would disclose the original trace and violate our privacy requirements. In the following, we thus outline a secure ShuffleRelease protocol that blindly shuffles trace fragments before releasing them.
The basic idea of ShuffleRelease is to let the users break up traces $T_i, ..., T_n$ into small encrypted fragments $[T_{i,1}], ..., [T_{i,n}]$ (in addition to outsourcing them in the hash table format for the OPSI protocol) which are first shuffled blindly by the cloud peers with the fragments of the other traces in the aggregate before all fragments are decrypted and released together. Intuitively, the blind shuffle prevents the cloud peers and any outside attacker from determining which fragment belongs to which trace $T_i, i, j \in A$.

We emphasize that the fragment length should generally be minimized to decrease the chances of an attacker to successfully recreate traces from the shuffled fragments. However, fragment length must also consider the respective use case: Our crowd-sensing campaign for elevation profiles (Sect. 5.4.4.4) requires fragments of minimum length two, because each reported measurement corresponds to the difference in the barometric pressure between two locations. In contrast, measuring people density [BBC12] is possible over fragments of length one, since each measurement simply reports the number of people at a discretized location or region. Being able to shuffle and release arbitrary fragments is an important mechanism since it tailors equally to applications which require data that is measured at single locations (e.g., noise levels [MSNS09] or people density [BBC12]) and to applications which require data to be measured continuously over multiple locations (e.g., map creation [OSM17] or road condition monitoring [MPR08]).

Figure 5.6 provides an example of our idea of shuffling trace fragments: Trace $T_1$ is broken up into fragments $\{(1, 2), (2, 3)\}$ which are encrypted individually by $U_1$. In the same way, users $U_2$ and $U_3$ prepare $T_2$ and $T_3$, respectively. Having determined the aggregate $A_1 = \{1, 2, 3\}$, the cloud peers put the corresponding fragments into a single big set and shuffle it blindly. Finally, the cloud peers release the shuffled set, e.g., $\{(2, 5), (1, 2), (3, 6), (2, 3), (2, 4), (1, 3)\}$. From this set of fragments, it is possible to build many traces besides $T_1, T_2,$ and $T_3$, e.g., $(4, 2, 3), (4, 2, 3, 1), (1, 2, 5)$ and so forth. Since the original traces are hidden among these numerous other possible traces, an attacker can only guess which are the original traces – we evaluate the attacker’s chances of success in detail in Section 5.4.4.3.

We now explain how to realize a secure ShuffleRelease protocol. As a preparation (Step 3.1 in Figure 5.6), each user divides her trace $T_i$ into $n$ fragments $T_{i,1}, ..., T_{i,n} \subseteq T_i$ and encrypts each fragment with the public key $PK_1$ of cloud peer $C_1$ using a semantically secure cryptosystem $Enc$ such as the Elliptic Curve Integrated Encryption Scheme (ECIES). The encryptions $[T_{i,j}] \leftarrow Enc_{PK_1}(T_{i,j})$ are then sent to the second cloud peer $C_2$. When an aggregate of $k$ intersecting traces is found in Step 2, $C_2$ shuffles the corresponding encrypted fragments under a secret permutation $\pi_2$ and sends them to $C_1$. $C_1$ shuffles all fragments again under its own secret permutation $\pi_1$ then decrypts and releases each fragment.

ShuffleRelease realizes a blind shuffle, i.e., no single cloud peer or outside party knows or is able to reverse the shuffle $\pi_1 \circ \pi_2$ in order to recreate the original traces from the released trace fragments. It is actually a much simplified version of our shuffle protocol in CoinParty (Chapter 3) for the semi-honest model stripped of the verification mechanism that is required only in the malicious adversary model assumed in CoinParty because of the involved monetary values.
5.4.3.4 Attacker

In line with prior works [HGXA07, HG05, STT+12, FSH09], we consider tracking as the primary attack on privacy. Being able to track a user along her trace, an attacker can infer private information and often even de-anonymize the user based on her unique mobility patterns [GH05, Kru07, GKdPC14]. The attacker’s success is usually measured in the distance over which the attacker can correctly track a user.

Before we explain in detail the attack, we make different worst-case assumptions: First, we assume that the attacker has background knowledge about each user. Concretely, we build mobility profiles for each user that are available to the attacker (cf. Figure 5.6, right). The mobility profiles tell the attacker the probability that user \( u \) moves from node \( x \) to \( y \). Second, we assume that the attacker knows the starting points of all users, e.g., a known home address [GP09]. Finally, we defensively assume that released fragments overlap such that they can be connected to traces. Our tracking attack would need adjustments for non-overlapping fragments but, more importantly, be much less effective. In conclusion, the assumptions we describe strengthen the attacker and thus yield a very defensive measure of the achieved privacy.

Under the given assumptions, the attacker tracks each user as follows: The attacker knows the starting point and follows the user until the first mix zone, i.e., an intersection of at least two traces, by following the fragments’ overlaps. A mix zone is simply a node \( n \) which is traversed by multiple users (e.g., node 2 in Figure 5.6). Thus, to track users across mix zones the attacker needs to guess which exit node \( e_1, \ldots, e_m \) was taken by which user. We use the Bayes estimator proposed in [FSH09] to derive the probability that user \( U_i \) exits the mix zone \( n \) at \( e \) by

\[
P_n(U_i|e, n) = \frac{P_n(e|U_i) P_n(U_i)}{\sum_{U_j \in U} P_n(e|U_j) P_n(U_j)}
\]

(5.3)

where \( P_n(U_i) \) and \( P_n(e|U_i) \) are derived from users’ mobility profiles. Applying Equation 5.3, the attacker computes the likelihood of all combinations of users and exit events. The attacker now computes a maximum weight matching between users and exit nodes using the likelihoods as weights, which results in the most probable assignment of users to exit events. Finally, we measure for each user how far along her contributed trace an attacker is able to correctly track her in the released aggregate following this assignment of exit events. We then define our overall privacy metric as the fraction of all users the attacker can track over a certain distance.

5.4.4 Evaluation and Discussion

We quantitatively evaluate performance, utility, and privacy of TraceMixer and conclude with a short qualitative discussion.

Implementation. We implemented a complete prototype of TraceMixer: We realize the user-side component as an Android application and the anonymization mechanism as well as the attacker in Python. For the performance-critical 2-OPSI-CA\((^m_2)\) scheme at the core of ObliviousAggregation, we extend the efficient C++ implementation of 2-PSI provided as part of the ABY stc framework [DSZ15].
Public-key cryptography for the trace preparation on the client-side and for the release step on the cloud peers is implemented using libsodium\(^4\). As encryption scheme for ShuffleRelease, we use the ECIES.

**Experimental setup.** We measure the performance of the user-side component on an LG Nexus 5 smartphone (Android 4.4, Qualcomm Snapdragon 800 @ 2.26 GHz, 16 GB RAM) and execute the anonymization mechanism between two desktop machines (Ubuntu 14.04, Intel i7-4770 @ 3.10 GHz, 16 GB RAM) that communicate over a 1 Gbit/s LAN. All results are given as mean and standard deviation over 5 independent runs (we stopped at five runs due to the significant runtimes and low standard deviation). To improve readability, we omit the standard deviation when it is below 1% of the measurement.

**Datasets.** We evaluate all three components of TraceMixer, i.e., the user-side application, the oblivious anonymization mechanism, and the attacker, on three (popular) real-world datasets: i) the Trucks dataset containing the trajectories of a fleet of Trucks in the city of Athens [FGPT05], ii) a subset of the Geolife dataset containing everyday human mobility [ZZXM09], and iii) a large dataset of sports activities we collected ourselves around the city of Aachen [Bav16]. We choose these datasets because they feature largely different characteristics (as summarized in Table 5.2) that allow us to evaluate TraceMixer in very different settings.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Number of traces</th>
<th>Number of users</th>
<th>Number of nodes</th>
<th>Mean nodes per trace</th>
<th>Timespan in days</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trucks</td>
<td>276</td>
<td>50</td>
<td>61,181</td>
<td>443</td>
<td>303</td>
</tr>
<tr>
<td>Geolife</td>
<td>1341</td>
<td>96</td>
<td>472,052</td>
<td>352</td>
<td>1631</td>
</tr>
<tr>
<td>Aachen</td>
<td>5229</td>
<td>3410</td>
<td>7,219,605</td>
<td>1,381</td>
<td>101</td>
</tr>
</tbody>
</table>

Table 5.2 Overview of the datasets used in the evaluation: We select datasets with different backgrounds and characteristics.

\(^4\)https://libsodium.org (accessed 2017-06-07)
Oblivious aggregation. The runtime and communication overheads of our OBLIVIOUSAGGREGATION protocol on the three datasets are shown in Figure 5.7. We process traces in their chronological order and measure the total runtime. TraceMixer requires only few minutes on the small Trucks dataset while runtimes range in the order of a few hours for the much larger datasets, Geolife and Aachen. Communication overheads range in the order of GBs to TBs depending on the size of the dataset. These significant overheads are due to the repeated invocations of 2-OPSI-CA($m^2$), e.g., in the most expensive case ($k = 50$ on the Aachen dataset) we require a total of 21,005 invocations of 2-OPSI-CA($m^2$) to anonymize the complete dataset.

We emphasize that all overheads are completely shouldered by the cloud peers – due to the repeatability property of our OPSI-CA scheme, users can go completely offline after they have once outsourced their traces as no further interaction is required. Although the overheads are significant even for the (unconstrained) cloud peers, they are manageable even if datasets are anonymized in one batch: For all three datasets, the presented runtimes for anonymization are less than 0.001% of the timespan over which the datasets were collected (cf. Table 5.2) such that cloud peers can anonymize traces on the fly as they are submitted.

Shuffle and release. Finally, we measure the performance of SHUFFLERELEASE on aggregates with up to 250,000 trace fragments. This corresponds to the number of fragments in our largest evaluation setting, i.e., an aggregate of $k = 50$ traces where each trace has a length of 5000 nodes (4747 is the maximum length in the three datasets). The overheads are linear in the number of fragments and the overall protocol is efficient. Even for the largest aggregates of 250,000 fragments the protocol finishes in 10.64s ($\sigma = 0.32s$) and communicates only 36.25 MB ($\sigma = 0.02$ MB). Thus, the overheads for shuffle and release are clearly feasible on the desktop machines that we run the cloud peers $C_1$ and $C_2$ on in this experimental evaluation.

5.4.4.2 Utility Evaluation

We discuss data utility along three criteria: i) spatial accuracy, ii) temporal accuracy, and iii) suppressed data.

Spatial accuracy. The most important utility criterion is the spatial accuracy of the data (cf. Section 5.4.2). In our approach, only the discretization step (cf.
Section 5.4.3.1) introduces small spatial inaccuracies but may also correct errors. Since discretizing locations is not the focus of this work, we did not evaluate these effects. Instead, we set a small discretization threshold of 2 m which introduces only little inaccuracies in comparison to the error of GPS receivers of today’s smart phones [Zan09]. Notably, our approach does not use spatial obfuscation to anonymize traces other than most related works. Besides the tunable inaccuracy due to discretization, TraceMixer thus achieves maximum utility in terms of spatial accuracy.

Temporal Accuracy. Traces uploaded to TraceMixer with timestamps. Further temporal information is stripped off to prevent reconstruction of individual traces through temporal correlation of the nodes, i.e., TraceMixer only releases the timespan of the traces in a released aggregate. Thus, temporal accuracy increases when aggregates are filled up faster. Figure 5.8(a) plots the average timespan per aggregate normalized by the timespan over which the dataset was collected, e.g., a timespan of 50% means that on average each aggregate is filled and released after half of the time it takes to collect the whole dataset. As we expect, a smaller privacy parameter $k$ leads to smaller timespans since aggregates are filled up faster. How fast aggregates of given size $k$ are filled then depends almost entirely on how fast new traces are uploaded by users which is not limited by our approach but determined by the temporal density of the collected dataset. In conclusion, temporal accuracy is coarse but still practical for our targeted scenarios (cf. Section 5.4.2).

Suppressed data. A final criterion of utility is the percentage of data that could be anonymized versus the percentage of data that had to be suppressed because the privacy criterion could not be fulfilled. As Figure 5.8(b) shows a non-negligible fraction of traces could not be anonymized because TraceMixer could not find suitable aggregates. Importantly, the majority of these traces were collected towards the end of the collection period. Since we process traces in chronological order, these traces are anonymized last and thus suitable aggregates often have already been released and new ones have not yet filled up. Thus, when data collection continues, the majority of these traces will eventually be released.

5.4.4.3 Privacy Evaluation

We evaluate the attacker’s success in reconstructing users’ traces from the released aggregates. For the attacker’s background knowledge, we construct mobility profiles
5.4. Case Study: Crowd-Sensing

Figure 5.9 Attacker’s tracking success on the three different data sets.

from on up to ten randomly selected traces per user. We repeat the attack 30
times. Figure 5.9 plots the cumulative distribution of the attacker’s success chances
measured as the fraction of users (y-axis) tracked over a given fraction of their whole
traces (x-axis) for privacy parameters $k = 5, 25, 50$ (lines). We do not plot error bars
since standard deviations are fairly low, i.e., 2.07%, 0.72%, and 0.31% on Trucks,
Geolife, and Aachen, respectively.

On all three datasets, the attacker quickly loses track of users, e.g., even on Geolife
(where the attacker is more successful than on the two other datasets) he can only
track 20% of the users longer than 20% of their traces for $k = 25$. Furthermore,
increasing the privacy parameter $k$ also significantly decreases the distance the at-
tacker can track users, i.e., privacy increases. As we observed in our performance
and utility evaluation, increasing $k$ only slightly increases runtime (Figure 5.7, left)
and has only a moderate impact on utility (Figures 5.8(a) and 5.8(b)). Thus, it is
clearly feasible and advisable to choose higher values for $k$.

5.4.4.4 Use Case: Crowd-Sourced Elevation Profiles

To present a concrete use case for TraceMixer, we realize a privacy-preserving crowd-
sourcing campaign for the creation of high-precision elevation profiles [Ind14]. GPS
altitude information is error prone and altitude profiles are often unavailable or
very coarse, especially in rural regions. In contrast, most modern smart phones are
equipped with barometers which measure altitude very precisely. We observed in
preliminary experiments that the precision of barometric altitude is within centime-
ters of the real altitude while GPS altitude may deviate by several meters.

To employ TraceMixer in this setting, we implement the client-side component as
an Android application which samples air pressure. The application runs as a back-
ground task every two minutes to minimize energy consumption. We let users report
only the difference in air pressure between two nodes. This avoids computing al-
titude from absolute air pressure measurements, which can vary significantly over
time due weather conditions. In contrast, the difference in air pressure between two
locations is much less impacted by local weather.

We distributed the application to nine voluntary users who collected a total of 3990
air pressure measurements anonymously through TraceMixer. Starting from one
reference point, we iteratively calculated the altitude of surrounding nodes. To quantify the error, we compare against the altitude data obtained from the local land-registry which features an altitude resolution of 0.1 m over a 0.2 m × 0.2 m grid. For TraceMixer, we observe an average error in the altitude of 0.99 m (σ = 0.91 m). This error is higher than our preliminary experiments suggest due to the very low sample frequency which leads to large distances between two sampling locations. Increasing the sample rate increases energy consumption at the client but allows us to significantly decrease this error.

5.4.4.5 Discussion of Security and Privacy Guarantees

Security guarantees. The core of TraceMixer is the oblivious aggregation which is based on our OPSI scheme that we proved secure in the semi-honest model in Section 5.3.4.3. A second important aspect of the security of TraceMixer is the security of ShuffleRelease. It is important to show that neither privacy peer can reverse the shuffle as this would allow them to recreate the original traces. For $C_1$ this is impossible, since it obtains the fragments already in completely random order from $C_2$. For $C_2$ this is impossible as well, since i) it learns nothing from the fragments due to encryption and ii) cannot relate encrypted fragments to decrypted fragments released by $C_1$ due to $C_1$’s own shuffle and the semantic security of the employed encryption scheme.

Privacy guarantees. It is important to note that the privacy parameter $k$ is strongly correlated to the actual achieved privacy level (Figure 5.9) but cannot guarantee that the attacker’s chances are below a certain threshold. Ideally, cloud peers would measure the achieved privacy obliviously and release aggregates only when the attacker’s success probability falls below a set threshold. However, measuring privacy over encrypted traces is computationally too expensive. Thus, cloud peers need to decrypt aggregates before they can measure the actual level of privacy. Still, they can decide to wait for and obliviously add further traces if the aggregate does not achieve the required privacy level. Though, this is not secure against an inside attacker compromising one of the cloud peers, it fully protects against outsiders. Additionally, the achieved privacy can be decreased through Sybil attacks: Without any countermeasures, an attacker can submit $k - 1$ traces that all intersect. The first real user who intersects with the fake traces can then be trivially tracked by the attacker. Protection against such Sybil attacks can be achieved in TraceMixer by requiring user authentication and ensuring that each trace in an aggregate is contributed by a different user.

5.4.5 Summary

We presented our first case study to test the applicability, scalability, and composit-ability of our 2-OPSI-CA($m$) scheme within the context of more complex systems. To this end, we proposed TraceMixer, a novel LPPM tailored to the special requirements in crowd-sensing (i.e., providing trajectory privacy protection while preserving high spatial accuracy) that hitherto remained unaddressed by prior works. With TraceMixer, we achieve these goals through an anonymization mechanism that is
inspired by and at the same time reinventing the concept of mix zones. Departing even further from existing approaches, TraceMixer renounces all centralized design patterns that make previous LPPMs vulnerable to attacks, leaks, and access by governments. Instead, TraceMixer’s core is implemented by composing our 2-OPSI-CA($m^2$) with further anonymization mechanisms, allowing TraceMixer to obliviously establish the notion of k-anonymity on traces.

As a thorough evaluation on three real-world datasets shows, our approach is feasible even for large datasets and introduces minimal spatial distortion while effectively protecting users’ privacy. Employing OPSI to outsource all heavy computations to two untrusted cloud peers, we reduce overheads for the mobile users to a minimum and enable them to contribute to the crowd-sensing campaign. To follow up on our evaluation, we carried out a crowd-sensing campaign through TraceMixer, which demonstrates the privacy-preserving creation of precise altitude profiles.

To conclude, TraceMixer provides a practical LPPM for a variety of crowd-sensing campaigns ranging from map creation [OSM17, GPA+10] over environmental monitoring [MPR08, MSNS09] to commercial applications [BBC12] and city planning [Str14]. This case study shows that our 2-OPSI-CA($m^2$) scheme is practical for constrained environments, can be composed with further non-trivial functionality, and scales to real-world problem sizes.

5.5 Case Study: Genetic Testing

As the third main part of this chapter, we present our second case study to validate the applicability, scalability, and adaptability of our OPSI schemes for the single-party outsourcing scenario. As motivation for this case study, we set out to solve the 2016 iDASH Secure Genome Analysis Competition [iDa16] (Section 5.5.1). In this competition, participants were challenged to securely outsource the computations for finding single nucleotide polymorphisms (SNPs) in a database of patients’ genomes to a single untrusted cloud, e.g., to test susceptibility for genetically determined diseases (Section 5.5.2).

As solution to the posed challenge, we propose our Bloom filter-based Outsourced Oblivious Matching (BLOOM) system [ZPH+17] which reduces the required database match to the set intersection cardinality problem that we securely outsource to the cloud through our k-OPSI($1^1$) scheme or, optionally, using the optimized but slightly insecure 2-OPSI-CA($1^1$) variant (Section 5.5.3). We implement, evaluate, and compare both approaches on the datasets provided by the competition organizers (Section 5.5.4). Fhe-BLOOM, our approach based on k-OPSI($1^1$), performs a disease susceptibility test on a database with 50 patients each with up to 100000 SNPs in approximately 5 min. Phe-BLOOM, our second approach based on the optimized 2-OPSI-CA($1^1$) variant, notably decreases runtimes by four orders of magnitude down to 75 ms. Fhe-BLOOM finished runner-up in the 2016 iDASH competition among submissions from eight other teams including Microsoft Research and IBM [WJT+17, HTJ16] - Phe-BLOOM was developed for the main publication [ZPH+17] after the competition and was not ranked (Section 5.5.5).
5. Outsourced Private Set Intersection

5.5.1 Motivation

Technological advances have made whole genome sequencing fast, accurate, and affordable for the masses [ACHT13]. Already today, several public initiatives [CV15, Chu05, GBH+03] have built large cohorts of volunteers willing to share their genomes to accelerate biomedical research. Meanwhile, an increasing number of private players offer services related to genomic data, e.g., tracing ancestry [23a]. Evidently, whole genome sequencing is here to stay and the massive collection, storage, and processing of human genome data has already become a reality.

On the other side, the genome era also brings unprecedented risks for personal privacy. Genomic information uniquely identifies its owner [HSR+08] and may be misused, e.g., for surveillance [NA99]. The genome further carries information about an individual’s appearance, health, or predispositions [JO09, ADL08] which could cause genetic discrimination. This is aggravated by the fact that genomes remain almost stable over time and, thus, cannot be revoked or replaced once leaked or made public [ACHT13, BBH+16]. Since relatives share large fractions of their genomes, an individual’s decision also affects the privacy of others (termed kin genomic privacy in [HAHT13]). Finally, the full extent of personal information that can be extracted from a person’s genome is still unknown and so are the associated privacy risks, e.g., whether it is possible to even predict human behavior from genomic analysis [Can07].

These significant personal privacy risks are aggravated by various attacks that have proved traditional anonymization mechanisms ineffective for genome data [Ma05]: Wang et al. [WLW+09] reidentify individuals in a genome-wide association study and apply their attack to the HapMap project [GBH+03]. Sweeney et al. [SAW13] use public demographics to re-identify a significant fraction of public profiles of the Personal Genome Project [Chu05]. Shringarpure et al. [SB15] re-identify individuals in public genomic data-sharing beacons.

In response to the failure of traditional methods, current research focuses on cryptographic techniques to protect genomic privacy [NAC+15, EN14]. In this context, SMC enables two relevant scenarios: i) Secure collaboration: two or multiple parties collaborate on their joint data, yet without disclosing their individual datasets. ii) Secure outsourcing: one or more parties outsource storage and processing of genome data to an untrusted cloud which remains oblivious of the data and computed analysis. In these settings, privacy-preserving variants have been proposed for genome-wide association studies [KL15b, CTW+15, ZBA15], sequence comparisons [KL15b, ZBA15], sequence alignments [ZMR+17, FDH+11], and disease testing [DCFGT12, ARM+13].

The applicability of protocols in the secure collaboration setting is, however, limited by their significant processing, communication, and storage overheads. Scalability issues are exacerbated by the typically huge amounts of data in genomics. Secure outsourcing is thus a promising approach to overcome these challenges by harnessing the elastic storage and processing resources in the cloud. In order to understand the limitations and potential of secure computations, especially secure outsourcing, the center for iDASH [ida] organizes yearly competitions to assess the state-of-the-art and stimulate further progress. The outcomes of the previous two competitions held in 2014 and 2015 are summarized in [JZW+14, TJW+16], while the starting point and motivation for this case study is the secure outsourcing task posed as part of the 2016 edition of the iDASH Secure Genome Analysis Challenge.
5.5.2 Problem Statement

The 2016 edition of the iDASH challenge comprises three tracks [iDa16] addressing: 1) privacy-preserving genomic data sharing, 2) secure sequence comparisons in the two-party setting, and 3) secure matching in the outsourcing setting. Before we present our two solutions to Track 3, we first concisely define the problem scenario and analyze relevant related work.

As illustrated in Figure 5.10, we assume a researcher who owns a database $D$ containing $m$ patient records $D_1, \ldots, D_m$. Each record comprises up to $n$ SNPs in variant call format [DAA+11]. The researcher also holds a query $Q$ of different SNPs and wants to obtain a list of those patients in the database that match all SNPs in the query. Since the data owner is not capable of analyzing the data locally due to limited computation and storage resources, she needs to delegate storage and processing to the cloud. Due to the sensitive nature of genomic data, the cloud must remain oblivious of the stored genomic data and all computations on it.

In this setting, the goal of the competition is to design a secure outsourcing scheme based on HE that allows the researcher to accurately, efficiently, and securely delegate computations and storage to a single cloud server. Importantly, a two-rounds protocol is required, i.e., after an initial setup phase the data owner poses the query and receives back the result without any intermediate interaction with the cloud service. The following are the detailed requirements set forth by the competition organizers for the solution of this problem [iDa16].

Accuracy: The researcher aims to find exact matches, i.e., a patient matches the query iff. all queried SNPs are contained in his patient record. The final result is a binary vector that indicates for each patient whether the query matches or not. Moreover, solutions are also judged by their ability to generalize, e.g., to fuzzy queries or partial matches.

Performance: The main optimization criterion is the query completion time which includes i) preprocessing and encryption by the researcher, ii) computations on the encrypted data in the cloud, and iii) postprocessing the final results by the researcher. Communication and processing overheads of the postprocessing step are tightly limited to 20 SNPs and 100 comparisons, respectively, for a database of $m = 50$ patients and queries of at most five SNPs. Communication
and storage overheads should be minimized but are considered only secondary optimization goals. Furthermore, overheads related to preparing and uploading the patient database should be reasonable one-time preprocessing overheads that are expected to amortize over multiple subsequent queries.

Security goals: Solutions must be secure in the semi-honest adversary model. All cryptographic primitives must offer at least 80-bit symmetric security or equivalent. The cloud must remain completely oblivious of the outsourced data and the results of the query. In particular, the length of the results must not leak the number of found matches. Furthermore, no access patterns should be leaked. The latter requirement was judged qualitatively and could be relaxed.

5.5.2.1 Related Work

From a technical perspective, BLOOM is closely related to PSI and OPSI, which we have already analyzed extensively before in Section 5.2.2. We explicitly refer to [WJT+17, HTJ16] for a summary of the results of the 2016 iDASH competition, which also quantitatively compares our solution to the other submissions.

The broader scope of our work is secure (outsourced) genome analysis. In the related literature, many privacy-preserving variants of applications with a genomics context have been proposed. Their focus has been on genome-wide association studies [KL15b, CTW+15, ZBA15, LYS15, KBLV13, ZDJ+15], sequence comparisons [KL15b, ZBA15], sequence alignments [ZMR+17, FDH+11], and statistical genomic tests [DCFGT12, ARM+13, DDC14]. Like ours, some of these works target the secure outsourcing setting [KL15b, LYS15, ZDJ+15] and make use of different flavors of HE. The majority of proposals in related work [CTW+15, ZBA15, ZMR+17, ARM+13, KBLV13] are set in the secure collaboration setting and can only be outsourced to two or more non-colluding cloud servers (resembling the security model in our multi- or two-party outsourcing setting). This would require a relaxation of the security requirements set forth in the iDASH 2016 challenge, as it introduces the additional security assumption that the two or more parties do not collude.

5.5.3 Secure Outsourced Queries over Genomic Data

We now present our two approaches, FHE-BLOOM and PHE-BLOOM, for the secure outsourced genetic disease testing scenario set in Track 3 of the 2016 iDASH challenge. Our basic approach is to efficiently represent the SNPs in the patients database and in the query as set elements. The posed question whether a patient matches all SNPs in the given query can then be simply answered by computing the cardinality of the set intersection between the query and each individual patient record and comparing it against the number of SNPs in the query – this only requires the data holder to remember the query but allows the data owner to fully forget the patient database once it has been outsourced. To compute the cardinality over securely outsourced datasets in the cloud, we employ our k-OPSI-CA(1) (in FHE-BLOOM) or 2-OPSI-CA(1) schemes (in PHE-BLOOM) presented in Section 5.3.3.

Figure 5.11 provides a combined overview of both approaches, FHE-BLOOM and PHE-BLOOM. We distinguish a preprocessing phase (upper part) during which the
patient database is encoded, encrypted, and uploaded to the cloud and an online phase (lower part) that comprises all steps required to process a query. On the highest level, our goals are i) to securely outsource as much processing as possible from the data owner (left) to the cloud (right) and ii) to minimize the online overheads for the complete processing of a fresh query.

Both approaches proceed in the following steps: At the beginning of the preprocessing phase (top part of Figure 5.11), the data owner holds a patient database with \( m \) patient records, \( D = \{ D_1, \ldots, D_m \} \). The data owner creates one empty Bloom filter \( B_i \) per patient record \( D_i \) (using keyed-hashing in \( \text{Phe-Bloom} \)) and inserts each patient’s SNPs (Step 1). The data owner then encrypts the resulting Bloom filters bitwise before she uploads and stores them securely in the cloud (Step 2). In \( \text{Fhe-Bloom} \), encryptions are packed row-wise, i.e., all \( l \) bits of a single Bloom filter \( B_i \) are packed and encrypted in only \( \lceil l/s_F \rceil \) ciphertexts. In contrast, we pack bits column-wise for \( \text{Phe-Bloom} \), i.e., all \( m \) bits of a column \( B'_i = (b_{1,i}, \ldots, b_{m,i})^T \) are packed into \( \lceil m/s_P \rceil \) ciphertexts. Note that these steps represent exactly the preprocessing and setup steps of our single-party OPSI schemes (Protocols 5.3 and 5.4 in Section 5.3.3). After these steps, the data holder can completely delete her copy of the patient database in order to free local storage and memory.

At the beginning of the online phase (bottom part of Figure 5.11), the data owner holds a fresh query \( Q \) to be matched against the database. In \( \text{Fhe-Bloom} \), we treat the query exactly the same as the datasets \( D_i \), i.e., the data owner transforms \( Q \) into a Bloom filter \( B_Q \) which she encrypts and uploads to the cloud (Steps 1 and 2).
In Phe-Bloom, being based on the specialized 2-OPSI-CA scheme, the query (the second dataset in Protocol 5.4) is transformed into a Bloom filter using keyed hashing and uploaded to the cloud without additional encryption (only Step 1). After these steps, the data holder may forget the query and needs to remember only the number of set bits in $B_Q$, denoted $|B_Q|$, to later check whether the query fully matched a patient record.

When the cloud service receives a fresh query, the cloud computes $m$ parallel runs of OPSI-CA in order to compute the cardinality of the intersection between the query and each patient record individually (Step 3). Note that the 2-OPSI-CA used in our second approach Phe-Bloom is optimized for parallel invocations and thus exactly fits this scenario.

In the final step, the cloud returns the encrypted results that contain the number of intersecting Bloom filter bits to the data owner. The data holder decrypts the result vector (Step 4) and checks in a simple postprocessing step whether the number of intersections between the query and patient record is equal to the set bits in the query Bloom filter $|B_Q|$ (Step 5). This check finds all correct matches but may produce false matches with a probability that is upper-bounded by the tunable false positive probability of the underlying Bloom filters. We also emphasize that the estimation error of the true set intersection cardinality (cf. Section 5.3.4.1) has no effect in this application of our OPSI-CA schemes since we do not compare the set intersection cardinality but only the set bits in the Bloom filter.

### 5.5.4 Evaluation and Discussion

We discuss and compare the performance of Fhe-Bloom and Phe-Bloom. We deliberately analyze and compare only our own two approaches since a rigorous comparison of Fhe-Bloom with the approaches of the other competitors approaches has been presented by the organizers of the iDASH challenge in [WJT+17,HTJ16].

We first formally analyze runtime and communication complexity (cf. Table 5.3), showing that both approaches scale linearly in the number of patients $n$ and the number of SNPs $m$ during setup while Phe-Bloom has a better complexity during the query phase. We then implement both approaches to thoroughly quantify their runtime, communication, and memory overheads. First, we benchmark both approaches using the evaluation setup of the iDASH competition [iDa16] (cf. Table 5.4). Afterwards, we conduct a more extensive evaluation of relevant parameters to study the performance of both approaches in greater detail. We conclude with a short security discussion.

#### 5.5.4.1 Complexity analysis

We compare the runtime and communication complexity of both approaches in Table 5.3. Following the evaluation criteria of the iDASH competition, we distinguish the following three phases: i) Database setup (Client) includes all steps required for preprocessing, encryption, and upload of the patient database; ii) Query (Cloud) comprises all computations by the cloud over encrypted data per query; iii) Query (Client)
### Table 5.3 Complexity analysis of Fhe-Bloom and Phe-Bloom

<table>
<thead>
<tr>
<th></th>
<th>Database Setup (Client)</th>
<th>Query (Cloud)</th>
<th>Query (Client)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time</td>
<td>Comm.</td>
<td>Time</td>
</tr>
<tr>
<td>Fhe-Bloom</td>
<td>$O(n/m)$ Enc$_F$</td>
<td>$O(n/m)$ C$_F$</td>
<td>$O(l/m)$ Mul$_F$</td>
</tr>
<tr>
<td>Phe-Bloom</td>
<td>$O(n/m)$ Enc$_P$</td>
<td>$O(n/m)$ C$_P$</td>
<td>$O(n/m)$ Add$_P$</td>
</tr>
</tbody>
</table>

The setup overheads are similar in both approaches and grow linearly in the number of patients $m$ and Bloom filter size $l$ which is proportional to the number of SNPs $n$, i.e., $l = -n \log(p)/\log(2)^2$. 

includes preparation, encryption, and upload of the query as well as download, decryption, and postprocessing of the result.

For simplicity, we measure runtime complexity in terms of the number of encryptions and decryptions as well as additions and multiplications in the encrypted domain. In comparison, (keyed) hashing causes only negligible overheads that are omitted in our complexity analysis. Communication complexity is measured in the number of exchanged ciphertexts, $C_F$ and $C_P$ for FHE and PHE, respectively.

**Database Setup (Client):** The setup overheads for both approaches scale linearly in the number of patients $m$ (the number of rows) and length of the Bloom filter $l$ (the number of columns). Fhe-Bloom and Phe-Bloom both support SIMD operations on the Bloom filter bits which decreases complexity by factors $s_F$ and $s_P$ denoting the number of slots of the FHE and PHE schemes, respectively. The exact values of $s_F$ and $s_P$ depend on the chosen key, e.g., $s_F = 1180$ and $s_P = 170$ in our evaluation. Longer keys provide higher security levels and more slots but also slow down en- and decryption as well as all operations over ciphertexts.

**Query (Cloud):** Fhe-Bloom requires the cloud to perform $O(n/l/s_F)$ additions and multiplications. In contrast, Phe-Bloom requires only $O(n(l/q)/s_P)$ additions which is orders of magnitude more efficient since it does not depend on the Bloom filter length $l$ that grows linearly in the maximum number of SNPs $n$ but instead on the query length $|q|$ which is a small constant in the competition scenario.

**Query (Client):** For Fhe-Bloom, the online processing and communication overhead per query for the client are in $O(l/s_P + m)$. Specifically, query preparation, encryption, and upload accounts for overheads in $O(l/s_F)$ while download and decryption account for overheads in $O(m)$. Phe-Bloom notably decreases the online overheads for the client by orders of magnitude to $O(n/m)$. This is achieved by the use of keyed hashing in 2-OPSI-CA(1) which obsoletes query encryption and thereby enables much more efficient query matching and a denser packing of the results. As we will discuss in detail later on, these significant complexity improvements of Phe-Bloom are made possible by tolerating a slight leakage of access patterns, while patient data stored in the cloud and computed results are still fully protected.

#### 5.5.4.2 Performance evaluation

We first describe our implementations of Fhe-Bloom and Phe-Bloom as well as the experimental setup then present a quantitative evaluation of runtime, commun-
Table 5.4 Evaluation results for the competition test cases (T1–T3) and asymptotic costs (As.)
in n: We measure i) database setup (preparing, encrypting, and uploading the database), ii) query processing in the cloud (matching query and database records in the encrypted domain), iii) query overheads on the client (pre- and postprocessing the query, including overheads for up- and download), and iv) total query overheads. Time is measured in seconds, memory and communication are measured in MBs. We fix $p = 2^{-14}$ for all benchmarks.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Database Setup (Client)</th>
<th>Query (Cloud)</th>
<th>Query (Client)</th>
<th>Query (Total)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fhe-Bloom</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T1</td>
<td>1</td>
<td>10000</td>
<td>5.73</td>
<td>91.8</td>
</tr>
<tr>
<td>T2</td>
<td>1</td>
<td>100000</td>
<td>35.24</td>
<td>105.8</td>
</tr>
<tr>
<td>T3</td>
<td>50</td>
<td>100000</td>
<td>1452.78</td>
<td>15.74</td>
</tr>
<tr>
<td>As. +1</td>
<td>1</td>
<td>100000</td>
<td>30.33</td>
<td>1.0</td>
</tr>
<tr>
<td>Phe-Bloom</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T1</td>
<td>1</td>
<td>10000</td>
<td>76.77</td>
<td>126.1</td>
</tr>
<tr>
<td>T2</td>
<td>1</td>
<td>100000</td>
<td>752.61</td>
<td>128.2</td>
</tr>
<tr>
<td>T3</td>
<td>50</td>
<td>100000</td>
<td>822.76</td>
<td>143.8</td>
</tr>
<tr>
<td>As. +1</td>
<td>1</td>
<td>100000</td>
<td>5.40</td>
<td>0.3</td>
</tr>
</tbody>
</table>

Implementation. Fhe-Bloom is implemented in C++ based on HElib [HS14, HE16]. All protocol steps are implemented in separate scripts that read inputs from and write outputs to disk. This allows us to process the encrypted database in chunks, which becomes necessary for the larger problem sizes to limit the memory footprint. Phe-Bloom is implemented in Python and uses our own implementation of the Paillier scheme [Pa19, DJ01] and msgpack⁵ for serialization. Despite being mostly unoptimized, our implementation of Fhe-Bloom manages to keep all data in memory for the problem sizes in our evaluation. However, chunk-wise processing of the patient database as in Fhe-Bloom is also straightforward to implement in Phe-Bloom if memory consumption needs to be reduced. Both implementations use the pybloom⁶ implementation of Bloom filters. For keyed hashing, we append a secret key to the hashed input just as done for salted hashes.

Experimental Setup. We perform experiments on a desktop client (Ubuntu 14.04 LTS, 8 cores at 3.40 GHz, 8 GB RAM) and a server (Ubuntu 14.04 LTS, 16 cores at 2.60 GHz, 32 GB RAM) that are connected by a 1 Gbit/s LAN. As in the competition, we limit execution on both machines to 4 cores. All cryptographic primitives are configured to offer at least 80-bit symmetric security according to NIST [BBB⁺07].

We use the datasets provided by the organizers of the iDASH competition and vary the parameters that determine the performance of our approach, i.e., the number of patients $m$, maximum number of SNPs $n$ per patient, and the Bloom filters’ probability $p$ for false positives. In order to reduce the parameter space for our evaluation, we fix $m = 10$, $n = 100000$, and $p = 2^{-14}$ and vary only a single of these parameters if

---

⁵http://msgpack.org/ (accessed: 2017-06-08)
not stated otherwise. Overheads for the database setup are measured over only 5 independent runs due to the higher runtimes compared to the online phase for which we execute 30 independent runs. We report the mean and the standard deviation (only when it exceeds 1% of the mean).

We use random matching and non-matching queries in our evaluation. However, we emphasize that for fixed \( n, m, \) and \( p \) the performance of our approach does not in any way depend on the number of queried SNPs and whether they match or not. Note that this is indeed a vital requirement for any solution as side-channel timing attacks become possible otherwise.

**Runtime**

We briefly discuss runtime overheads for the database setup and then focus on the primary optimization goal, the turnaround time per query in the online phase.

**Database setup.** Table 5.4 compares how both approaches perform in the three test cases of the iDASH competition [HTJ16, WJT+17]. We observe that overheads for preparing, encrypting, and uploading the database have the same order of magnitude for both approaches. Overheads of Fhe-Bloom are lower for \( m = 1 \) (Test 1 and 2) but grow roughly linearly in the number of patients \( m \) (Test 3). In contrast, Phe-Bloom has higher overheads first but, interestingly, these grow much slower when increasing \( m = 1 \) (Test 2) to \( m = 50 \) (Test 3). This is due to the different packing strategy of Phe-Bloom which encrypts Bloom filter bits column-wise and easily fits the \( m = 50 \) values per column \( B^j \) into the \( s_p = 170 \) slots of a single ciphertext.

To draw a fair comparison between both approaches, we estimate from our measurements the asymptotic runtime for outsourcing a single patient record with a fixed maximum number \( n = 100\,000 \) of SNPs (i.e., the time for adding the \( m + 1 \)th patient for large \( m \)). Fhe-Bloom then requires 30.33s \((\sigma = 1.48\,s)\) per patient in the database, while Phe-Bloom only requires 5.40s \((\sigma = 0.12\,s)\). While these overheads are non-negligible, they are still reasonable and need to be computed only once since the repeatability property of our OPSP-CA schemes allows an unlimited amount of queries on these datasets afterwards.

**Query turnaround time.** For Fhe-Bloom, the query runtime increases linearly in \( m \) and \( n \) and is in the order of seconds for Test 1 and 2 and in the order of minutes for Test 3 (cf. Table 5.4). Adding a single patient with \( n = 100\,000 \) SNPs asymptotically contributes only 5.52s \((\sigma = 0.29\,s)\) to the total query turnaround time. Notably, almost all of the asymptotic overheads are outsourced to the cloud. In comparison, Phe-Bloom’s asymptotic online overheads of 0.44ms \((\sigma = 0.025\,ms)\) are four orders of magnitude smaller.

Figures 5.12 and 5.13 (left) further break down the runtime overheads per query for different choices of \( m, n, \) and \( p \). The plots show mean and standard deviation of the runtime and communication for i) preprocessing the query on the client, ii) uploading the query, iii) processing the encrypted data in the cloud, iv) downloading, and v) postprocessing the results. The striking result for both approaches is that the cloud overheads (yellow bars) dominate the overall runtime which is clearly desired in outsourcing scenarios. We now discuss the runtime dependency of Fhe-Bloom and Phe-Bloom on the different parameters \( n, m, \) and \( p \) in more detail.
5. Outsourced Private Set Intersection

(a) The total query time of \textsc{Fhe-Bloom} grows linearly in $m$ while the overhead for the client remains almost constant.

(b) Communication of \textsc{Fhe-Bloom} grows linearly in $m$ while the initial constant overheads for the upload of the query dominate.

(c) The total query time of \textsc{Fhe-Bloom} as well as the client’s runtime grow linearly in $n$ (note the non-linear x-axis).

(d) Communication of \textsc{Fhe-Bloom} grows linearly in $n$ (note the non-linear x-axis) and is completely dominated by the query upload (the download overheads being barely visible at all).

(e) The total query time of \textsc{Fhe-Bloom} grows linearly with exponentially decreasing false positive probability $p$ (note the logarithmic x-axis) and is dominated by the cloud.

(f) Communication of \textsc{Fhe-Bloom} grows linearly with exponentially decreasing $p$ (note the logarithmic x-axis) and is completely dominated by the query upload (the download overheads being barely visible at all).

Figure 5.12 Runtime (left) and communication overheads (right) for \textsc{Fhe-Bloom} for increasing numbers of patients $m$ (upper), increasing maximum number of SNPs $n$ (middle), and exponentially decreasing false positive probability $p$ (lower).
5.5. Case Study: Genetic Testing

For FHE-Bloom, we observe linear growth in the number of patients \( m \) (Fig. 5.12(a)) and in the number of SNPs \( n \) (Fig. 5.12(c)) while runtime increases only logarithmically with decreasing false positive probability \( p \) (Fig. 5.12(e)). As desired for outsourcing, the cloud shoulders most of the runtime overheads in all cases.

For PHE-Bloom, we measure the same query turnaround time for all \( m \leq s_P = 170 \). This is expected since overheads of PHE-Bloom increase only stepwise every \( s_P \) patients because up to \( s_P \) patients are packed into a single ciphertext and processed in SIMD fashion. To confirm this behavior, we evaluate PHE-Bloom for numbers of patients that are multiples of \( s_P \), i.e., \( m \in \{10s_P, 20s_P, \ldots, 60s_P\} \) (Fig. 5.13(a)). Indeed, we observe a linear growth of the postprocessing overheads due to the increased number of decryptions. As for FHE-Bloom, the query turnaround time of PHE-Bloom grows linear in \( n \) (Fig. 5.13(c) and logarithmic in \( p \) (Fig. 5.13(e)). Notably, PHE-Bloom easily scales to thousands of patients and, as desired, most runtime overheads are shouldered by the cloud.

In summary, the runtimes of FHE-Bloom are within the order of minutes even for the largest parameter choices which we deem reasonable for practical deployments. In comparison, PHE-Bloom is at least three orders of magnitude faster and answers queries on a database with thousands of patients in milliseconds.

Communication

Table 5.4 shows the communication overheads for i) the upload of the encrypted patient database and ii) the upload of the query plus the download of the results. We first discuss overheads for the database setup and then focus on the more important optimization goal, the communication per query in the online phase.

Database setup. We observe that FHE-Bloom has lower overheads for a small number of patients \( m \) than PHE-Bloom since PHE-Bloom’s packing strategy pays off only for larger numbers of patients. To confirm this behavior, we measure the asymptotic communication overheads per patient in the database. For \( n = 100000 \), FHE-Bloom asymptotically uploads 265.29 MB while PHE-Bloom communicates only 3.13 MB per patient during setup of the database. Thus, communication overheads for the setup of large patient databases in PHE-Bloom are almost two orders of magnitude smaller than in FHE-Bloom.

Communication per query. Complementing Table 5.4, the right-side plots in Figure 5.12 and 5.13 further break down the communication overheads of FHE-Bloom and PHE-Bloom for different choices of the parameters \( m \), \( n \), and \( p \). The communication overheads of both FHE-Bloom and PHE-Bloom are dominated by the upload of the query which scales linearly in the number of SNPs \( n \) – the overheads for downloading the results are at least two orders of magnitude smaller and scale in the number of patients \( m \). Asymptotically, FHE-Bloom uploads a fixed 265.29 MB per query irrespective of \( m \) while downloading only 0.45 MB per patient in the database. PHE-Bloom uploads only 0.25 MB per query and downloads as little as 1.58 B (\( \sigma = 0.00 \) B) per patient which is only little more than the theoretical lower bound of \( \log_2(|Q| \cdot k) \) bits. In comparison, PHE-Bloom’s communication overheads are thus three orders of magnitude smaller than those of FHE-Bloom.
5. Outsourced Private Set Intersection

(a) The runtime for postprocessing in Phe-Bloom grows linearly in $\lceil m/s_P \rceil$ while the cloud’s runtime is almost constant.

(b) Communication of Phe-Bloom grows linearly in $\lceil m/s_P \rceil$ while the initial constant overheads for the upload of the query clearly dominate.

(c) The total query time of Phe-Bloom grows linearly in $n$ (note the non-linear x-axis) and is completely dominated by the cloud overheads.

(d) Communication of Phe-Bloom increases linearly in $n$ (note the non-linear x-axis) and is dominated by the query upload.

(e) The total query time of Phe-Bloom grows linearly with exponentially decreasing $p$ (note the logarithmic x-axis) and is dominated by the cloud.

(f) Communication of Phe-Bloom grows linearly with exponentially decreasing $p$ (note the logarithmic x-axis) and is dominated by the query upload.

Figure 5.13 Runtime (left) and communication overheads (right) for Phe-Bloom for increasing numbers of patients $m$ (upper), increasing maximum number of SNPs $n$ (middle), and exponentially decreasing false positive probability $p$ (lower).
5.5. Case Study: Genetic Testing

Memory

Table 5.4 shows the cloud’s and the client’s memory footprint during setup and query. For both approaches, the client’s memory consumption is fairly low in both phases with a maximum consumption of 156.86 MB and 143.85 MB in Fhe-Bloom and Phe-Bloom, respectively (in the third and largest test case, i.e., \( n = 100000 \) and \( m = 50 \)). Given that also the asymptotic memory consumption is low in both approaches, we consider these overheads manageable even for mobile clients.

For Fhe-Bloom, the cloud’s memory footprint of only 92.97 MB is fairly low as well. This is achieved by storing the encrypted database on disk and reading and processing it in chunks of the desired size. While this introduces perceivable I/O overhead, it becomes necessary for the larger problem settings, e.g., in Test 3 where the encrypted database size of 13.26 GB clearly exceeds the client’s 8 GB RAM.

In contrast, the encrypted database in our second approach, Phe-Bloom, is at least one order of magnitude smaller and can be kept entirely in memory. While this saves I/O overhead, it increases the memory footprint to 208.70 MB and asymptotically requires 12.25 MB. In the unusual scenario where the cloud has constrained memory, the database could also be processed chunk-wise to reduce memory overhead, similarly to our implementation of Fhe-Bloom. Overall, the cloud’s memory overheads are thus feasible in both approaches even for large problem sizes.

5.5.4.3 Security Discussion

In this section, we discuss our two approaches, Fhe-Bloom and Phe-Bloom, w.r.t. their security guarantees and their limitations and potential extensions.

The security of Fhe-Bloom follows directly from the security of the underlying k-OPSI-CA(1) as discussed in Section 5.3.4.3. In summary, the security of our OPSI-CA scheme guarantees that the cloud learns nothing about the content of the patient database, the query, or the results except for the number of patients \( m \) in the database and maximum number of SNPs \( n \) per patient (only when the Bloom filter’s false positive probability \( p \) is public). This especially implies that the cloud learns nothing about the queried SNPs and cannot even distinguish whether the same query was posed twice, i.e., no access patterns are leaked.

The security of Phe-Bloom follows from the security of the underlying 2-OPSI-CA(1) which leaks information about the posed query. In consequence, the cloud can distinguish whether queries are different or the same and if they overlap – due to the use of keyed hashing the cloud cannot, however, infer which SNPs are queried.

This slight leakage of access patterns might be tolerable in scenarios where high performance is paramount. Importantly, Phe-Bloom still fully protects outsourced patient database and computed results.

Phe-Bloom shows that tolerating a slight leakage of access patterns leads to significant performance speed-ups. To conclude our security discussion, we briefly point out a different relaxation of the security requirements that can achieve performance improvements. As we have noted in our analysis of related work, a significant part of the research on secure genomic analysis relates to the secure collaboration setting. In this setting, we have found that state of the art techniques (e.g., Yao’s GCs [Yao86],
the GMW protocol \cite{GMW87}, or secret sharing-based SMC \cite{CDM00}) impose lower overheads on the data holder than (fully) homomorphic encryption (cf. Figure 5.3 in Section 5.3.4.2). Unfortunately, these are interactive protocols that require collaboration of multiple parties and are thus inapplicable in a strict secure outsourcing setting. Relaxing the strict requirement of outsourcing to a single party would render these techniques applicable in the problem scenario of this case study. Instead of outsourcing to a single cloud provider, the data owner would securely split data across two cloud services and instruct them to carry out the genomic computations together using any of the established secure computation techniques. It is important to note that this relaxation introduces the additional security assumption that the two or more parties do not collude which must be further discussed in the context of genomics.

5.5.5 Summary and Future Work

The grave privacy risks and failure of traditional protection schemes make evident the need for strongest possible protection for genomic data which is currently achieved through SMC protocols that share data only in cryptographically protected form such that it is never learned by third parties in clear. The 2016 Secure Genome Analysis Competition, organized by the iDASH center, assesses the state of the art of these techniques to stimulate progress and foster adoption among non-experts. In this case study, we presented two solutions to Track 3 of this competition, the secure outsourced disease testing challenge.

Our core idea is to reduce the stated problem to PSI-CA and then build on our OPSI-CA schemes for secure outsourcing to the cloud. Our first approach, Fhe-Bloom, uses our k-OPSI-CA($^1$) which is based on fully homomorphic encryption. In our second approach, Phe-Bloom, we build upon our highly optimized 2-OPSI-CA($^1$). While this scheme slightly relaxes security guarantees, it reduces query turnaround times and communication overheads by two to four orders of magnitude. In the iDASH competition \cite{iDa16,WJT+17,HTJ16}, Fhe-Bloom was ranked runner-up to Microsoft’s submission, while Phe-Bloom was developed only afterwards but presents an exciting alternative when high performance is of paramount importance.

Our evaluation identified the setup of the patient database as the most expensive step in both approaches. These overheads may overtax constrained users who wish to outsource very large databases and should be further minimized in future work. However, due to the repeatability property of the underlying OPSI-CA schemes, these constant setup overheads amortize quickly over multiple subsequent queries, while the flexible data handling property ensures that data owners can add, modify, and delete records in the outsourced database without repeating of the entire setup phase.

A second important direction of future work is in answering different query types. While both Fhe-Bloom and Phe-Bloom are designed to compute exact matches, we further showed how to estimate the size of the intersection, e.g., in order to compute partial matches. In particular, this allows answering negative queries efficiently, e.g., which patients do not show certain variations. Both approaches also allow us to compute arbitrary linear combinations of the patients’ SNPs, e.g., in order to answer weighted queries for disease tests where certain SNPs are more critical
than others. Concretely, this can be implemented in both approaches by assigning integer weights instead of bits to the Bloom filter slots and interpreting the final results as the weights of the matching. Finally, with Bloom filters at the core of both approaches, a wide variety of corresponding extensions exist to add support for, e.g., range queries [AKL13] or locality-sensitive hashing for fuzzy queries [IM98].

5.6 Conclusion

Motivated by the important problem of securely computing set intersections and its increasing relevance in real-world applications, we pointed out different challenges for PSI arising in constrained applications scenarios. Motivated by the promising results obtained for outsourcing classifiers in the previous chapter, we investigated outsourcing schemes for PSI protocols as a promising approach to tailor PSI protocols to constrained applications scenarios, addressing our third research question (customization). We adapted existing PSI protocols to our outsourcing settings involving multiple or two cloud peers and designed novel efficient OPSI schemes for the single-party outsourcing scenario. We evaluate our OPSI schemes to show that overheads are indeed practical for the typical processing, communication, and memory constraints in mobile deployments.

To further validate the applicability, scalability, and flexibility of our proposed designs, we present two case studies which comprehensively study three of our proposed OPSI schemes within the context of more complex real-world applications. In particular, we propose the TraceMixer system [ZHBW17] for privacy preserving crowd-sensing as well as the BLOOM system [ZPH+17] for secure genomic disease testing which build on our OPSI schemes. Besides proving that we can effectively outsource set intersections securely from constrained data owners to untrusted computation clouds, these case studies demonstrate the importance of further properties we have set forth for practical OPSI schemes, i.e., repeatability, flexibility in data handling, and composability.
Conclusion

The need for data security and privacy protection in existing and emerging digital services increases [ZGMW14], among others, due to the huge success of data-driven business models [CML14, MC13], centralization of data in cloud services [CZ12, TJW+16], compromised end systems [Pri17, Thi16], illicit data processing not least for surveillance [Lyo14], and the impuissance of legislation to address these issues. We identified Secure Multiparty Computation (SMC) as a promising approach to provide rigorous protection against these threats with provably security and privacy guarantees.

The potential of SMC, however, is foremost on a theoretical level, as it is often deemed too inefficient and impeding in practice [ABPP16]. This thesis is motivated by the observed gap between the theoretical strength and potential of SMC and its feeble adoption in real-world applications. On the highest level, the goal of this thesis is thus to bridge this gap and contribute significantly towards the technical maturity of SMC and establishing it as a practical data security and privacy protection approach.

Our key motivation is the observation that SMC does not necessarily prevent but may even enable innovative digital services by helping to reconcile public, private, and business interests, e.g., in medical or financial applications that involve highly sensitive data.

Based on our qualitative and quantitative analysis of SMC frameworks [ZMHW15], we motivate three pressing research challenges that need to be overcome to reach our goals of establishing SMC in practice, i) extending the functionality (Q1), ii) increasing the efficiency (Q2), and iii) improving the customizability of SMC (Q3). Our guiding goal of establishing SMC as a practical protection technique motivates us to choose a use case-driven research methodology to address these questions. Identifying convincing use cases of SMC is an important (non-technical) research challenge in itself, but allows us to motivate and validate the practical applicability of all our technical contributions. The problem scope of this thesis is thus defined by i) motivating SMC through compelling real-world applications and ii) addressing the identified research questions in the course of the technical realization of the identified use cases.
6.1 Contributions

We address the aforementioned research questions in three distinct contributions. In the following, we first summarize the most prominent design aspects and results for each contribution. The core of all our contributions are improved, novel, or customized SMC protocols and frameworks that are of independent interest beyond the use cases considered in this thesis, e.g., they contribute generic functional building blocks to complement or improve other SMC frameworks.

6.1.1 Secure Decentralized Mixing of Digital Currencies

Motivated by the importance of anonymous payments as an instrument to exercise the right of informational self determination, we investigate financial privacy in decentralized cryptocurrencies by the example of the most wide-spread representative, Bitcoin. Previous work on de-anonymization shows the need for mixing services that help (potentially) de-anonymized users to re-establish their privacy. Many such mixing services have been proposed, but we identify shortcomings inherent to their design among all of them, e.g., weak security guarantees, limited anonymity sets, lacking means of plausible deniability, or insufficient scalability.

We propose CoinParty \([ZGH^{15}, ZMH^{16}]\), a novel efficient and decentralized mixing service that overcomes restrictions of prior approaches by modeling and realizing mixing as an SMC protocol for the first time. Implementing and evaluating CoinParty, we show that CoinParty i) offers strong security guarantees according to the general SMC security definitions and means to disincentivize and punish protocol-level Denial of Service (DoS) attacks, ii) affords up to two orders of magnitude larger anonymity sets than prior proposals, iii) provides plausible deniability, iv) scales to hundreds of users, v) minimizes transaction and mixing fees, and vi) remains fully compatible with the original Bitcoin system and easily accessible through standard software. To realize these advantageous properties, we emulate the insecure centralized mixing model securely through an SMC protocol which requires us to design a novel oblivious shuffle scheme, thereby extending SMC functionality (Q1).

Modeling mixing as an SMC problem provides us with tight upper bounds on the resistance against colluding peers, i.e., CoinParty tolerates up to \(\tau < m/3\) malicious peers. At first sight, such collusion of malicious peers seem to limit the security of CoinParty, but the following observations relativize and rebut this: First, the threshold \(\tau < m/3\) is optimal given our scenario and assumptions – stronger assumptions, such as the existence of a reliable broadcast channel, directly improve CoinParty’s collusion resistance. Second, other approaches [Max13a, RMSK14] achieve higher thresholds only through Bitcoin specific mechanisms – these cannot be proven secure in the SMC model and must thus be treated with care. Finally, we note that certain collusion could actually be desired, e.g., to build quorums of peers that are able to collaboratively de-anonymize users for the persecution of particularly serious crimes (distributing the quorum geographically affords resilience against local jurisdictions with too encroaching interests of inner security). Finding ways to reconcile Privacy Enhancing Technologies (PETs) with potentially opposing public interests is a research topic of growing importance.
6.1.2 Privacy-preserving Pattern Recognition and Classification

We motivate that emerging interactive applications of pattern recognition and machine learning are hindered due to the concerns of involved parties over sharing their sensitive input data or expensive classification models. We identify Secure Two-Party Computation (STC) as a promising approach to address these data security and privacy concerns, but find that the high processing requirements and data consumption of pattern recognition and machine learning challenges state of the art STC protocols, requiring us to improve their efficiency (Q2).

We propose SHIELD [ZVHW14, ZMR+17], an efficient STC framework for pattern recognition and machine learning upon which we realize different classification and pattern recognition approaches that overcome the identified security and privacy problems. In particular, we provide secure variants of Naive Bayes and Hyperplane classifiers, Artificial Neural Networks, as well as Forward and Viterbi computation on Hidden Markov Models. Our thorough evaluation shows that our approaches are accurate, secure, and outperform the state of the art in different network settings. Additionally, we demonstrate their applicability in three real-world use cases: indoor localization, spam recognition, and bioinformatics services.

Developing these use cases, we identify the need to customize SMC protocols to mobile application scenarios (Q3) and propose two solution strategies. We first present BOMA [ZHH+15], our bandwidth-optimized secure protocols for computing the maximum and its argument, an operation that is universally used in the design of our secure classifiers and many other STC applications, e.g., auctions [NPS99, KSS09], biometric identification [BCP13, EHKM11], or analysis of financial data [BTW12, BJSV15]. BOMA trades increased local processing against significantly reduced communication overheads, which even reduces overall runtimes in networks with limited bandwidth and, thus, presents a strong motivation to generally elevate communication overheads of SMC to a first-class optimization goal.

Despite the significant optimizations realized in BOMA, the remaining overheads of our secure classifiers may still overtax constrained mobile devices and networks. We thus propose a second orthogonal but fully compatible strategy. Building on our approach for outsourcing Forward computation [ZMR+17], we show how the secure classifiers in the SHIELD framework can be securely outsourced to harness the elastic resources of untrusted computation clouds. In our evaluation, we measure small processing and communication costs for the outsourcing steps that are feasible even for constrained devices and networks.

6.1.3 Outsourced Private Set Intersection

The results of the previous contribution motivated us to thoroughly investigate secure outsourcing to untrusted clouds as a general solution for tailoring SMC protocols to challenged environments (Q3). To this end, we concentrated on the problem of Private Set Intersection (PSI) since it is one of the most widely studied problems in the field of SMC. This allowed us to build on the multitude of existing traditional PSI protocols in order to fully concentrate on our primary goal of customizing SMC protocols to challenged deployment and operation scenarios.
As the first part of our third contribution, we presented three adapted and two novel Outsourced Private Set Intersection (OPSI) schemes for a comprehensive selection of usage scenarios, i.e., outsourcing to and from a single, two, or multiple parties. Our OPSI schemes show qualitative (in terms of security, composability, and flexibility) as well as quantitative (in terms of runtime and communication overheads) advantages over prior works. These OPSI schemes were developed as part of our TraceMixer\cite{ZHBW17} and BLOOM systems\cite{ZPH+17}, two independent contributions that we present in the second part of our third contribution as dedicated case studies to demonstrate and validate the real-world applicability of our OPSI schemes.

TraceMixer \cite{ZHBW17} is a novel location privacy protection mechanism tailored to the special requirements encountered in crowd-sensing. Our OPSI schemes provide TraceMixer with stronger privacy guarantees than prior location privacy mechanisms and allows a large numbers of mobile users to dynamically participate despite their constrained computation or communication capabilities.

The BLOOM system \cite{ZPH+17} realizes the outsourced testing of huge patient databases for genetic variations. Here, our OPSI schemes allow a single constrained user to harness the virtually unlimited storage and processing resources in the cloud. BLOOM was awarded second place in the 2016 Secure Genome Analysis competition \cite{iDa16} organized by the center for integrating Data for Analysis, Anonymization, and SHaring (iDASH).

6.2 Future Work

This thesis contributes to the technical maturity and real-world applicability of SMC by providing i) extended functionality, ii) increased efficiency, and iii) customization to challenged environments. Each of our contributions, as we summarized at the end of each chapter, opens up interesting research directions that will progress SMC technology depthwise. In order to fully establish SMC breadthwise as a practical protection mechanism, further challenges remain open that are significantly beyond the topical and temporal scope of this thesis. In the following, we briefly summarize the most important steps in the chronological order that we propose to address them.

Integrating SMC with other PETs: As noted in Section 2.2.2, SMC offers rigorous and comprehensive protection for the input but does not consider the privacy implications of the computed results. SMC must thus be integrated with other PETs to provide comprehensive protection. State of the art approaches to protecting the outputs of computations over sensitive data build on the differential privacy framework \cite{DMNS06,Dwo08}. Indeed, we observe various efforts to compute a differential private functionality securely using cryptographic methods, which are, however, so far restricted to rather simple aggregation tasks \cite{RN10,GXS13,BRB+17}. These works show that the combination of SMC and differential privacy is a powerful and far-reaching approach to data security and privacy that requires further investigation. In the context of this thesis, augmenting our SHIELD classification framework with differential privacy, e.g., protocols for securely training differentially private classifiers, is interesting future work.
New computing and collaboration paradigms: New computing, communication, and collaboration paradigms emerge continuously, e.g., from peer-to-peer networking [Wal03] to cloud computing [Gro09,TJA10], from Wireless Sensor Networks [CP03] to the Internet of Things [AIM10,ZGMW14], Crowd-sensing [GYL11], and Fog Computing [SW14]. In all of these paradigms, we observe that sensitive data is processed interactively between different parties between whom no prior trust exists. On the one hand, SMC naturally lends itself to enabling collaboration in these existing or emerging paradigms by guaranteeing data security and privacy to all involved parties. On the other hand, the traditional assumptions under which SMC is developed, i.e., high powered hosts connected over stable networks, is deeply opposed to the design philosophies and interaction patterns in these computing paradigms. Further work is necessary to make upcoming computation and collaboration paradigms accessible to SMC. In our third contribution, we showed how outsourcing SMC contributes to secure cloud computing and crowd-sensing – fog computing has similar traits and extending our contributions to fog computing is thus a natural next step.

Systematizing and standardizing SMC: One reason for the hesitant adoption of SMC in practice is the high entry barrier [KL15a, Chapter 3]. Standardization is the traditional way to decrease costs of adoption and becomes increasingly relevant for SMC in order to organize the many diverse techniques and applications that have been proposed so far. A possible way to approach standardization of SMC is to extract reoccurring usage scenarios and protection profiles and systematize these, e.g., Perry et al. [PGFW14] quantify the trade-offs in different SMC approaches and present a tool to support decision-making. As noted above, multiparty sum aggregation is a reoccurring pattern [CSS12,GXS13,CAF13,BRB+17] for which SMC-based solutions have even been adopted in practice [BIK+16,Kre17]. Combining frequent deployment and protection patterns with a systematization of SMC trade-offs and protection profiles presents a promising starting point for standardization efforts.

Anchoring SMC in legislation, regulation, and compliance: Current data protection legislation, regulation, and compliance centers around the fuzzy notion of personally identifiable information (PII). However, the ongoing digitalization and adoption of new information and communication technologies unlock and combine new sources of data that often enable unforeseen ways of identification and increasingly blur the line between PII and non-PII [Ohm09]. Already today, companies and public offices, thus, struggle in their efforts to comply with applicable data protection regulation – the upcoming EU General Data Protection Regulation (EU 2016/679), which couples fines for violation to a company’s sales, only aggravates these problems [var14]. SMC addresses data protection proactively by preventing that sensitive personal data is shared in the first place and thereby helps to overcome the inherent limitations of data protection legislation based on the fuzzy and increasingly outdated notion of PII. When a reasonable level of standardization is reached, SMC should, thus, be anchored in legislation as an approved protection technique, e.g., as a recommendation in the protection profiles issued by the German Federal Office of Information Security for certain application areas such as cloud computing, critical infrastructure, and identity management.
6.3 Concluding Remarks

In this thesis, we motivated the theoretical potential of SMC as a rigorous and proactive approach to data security and privacy and present a range of technical contributions that render SMC applicable in practice and significantly widen its scope of application. While SMC is not the panacea for data security and privacy protection, we have presented strong evidence that it is far more practical than commonly assumed.

Ultimately, it is the task of national and international policy to align public interests for safety with personal freedom and to reconcile the evermore data-hungry business models with personal privacy. The improved and novel SMC protocols and use cases presented in this thesis widen the technical design space for privacy-preserving data-driven digital services and show that the aforementioned goals, contradictory as they may seem, can actually be reconciled. To fully realize this potential of SMC in practice, our contributions and the general technical efforts in the field of SMC must be integrated with other techniques, considered for standardization, and ultimately anchored in legislation.

Our contributions have already been adopted and further improved by others and we hope that the identified directions of future work will stimulate further research in this important field.
Additional Evaluation of Selected Building Blocks in SHIELD

The measurements presented in this section complement the evaluation of our SHIELD framework by an evaluation of the secure building blocks, i.e., ScalarProduct, MaxArgmax, EvalProbMass, LogSum, and FunctionApprox. An overview of all building block and their utilization in the presented secure classifiers is given in Figure 4.5 in Section 4.4. For each primitive, we evaluate the numerical accuracy as well as the runtime and communication overheads.

A.1 Scalar Products

Scalar products are ubiquitous in machine learning and pattern recognition algorithms. In SHIELD, the secure ScalarProduct (Protocol 4.3, Section 4.4.2.1) is an important building block in the designs of Hyperplane and ANN. For our ScalarProduct, we batch the multiplication protocol on additive shares proposed by Demmler et al. [DSZ15] and extend it to our fixed-point number representation by applying Rescale at the end.

Accuracy. The numerical accuracy of ScalarProduct is exactly equal to that of Hyperplane, which we have already evaluated in detail in Section 4.4.8.1.

Runtime and Communication. We evaluate ScalarProduct on two vectors of growing sizes $n \in \{10^1, 10^2, \ldots, 10^3\}$. Apart from the negligible constant overhead for the single invocation of Rescale, the runtime and communication overheads of ScalarProduct are linear in the vector size. The overall overheads are very low, e.g., requiring only 20.02 ms ($\sigma = 0.27$ ms) and 0.92 MB for a scalar product on vectors of $n = 10^3$ elements.
A.2 Max and Argmax

Max and argmax are indispensable building blocks for classification algorithms. Our secure MAXARGMAX (Protocol 4.4, Section 4.4.2.2) is thus used in all of SHIELD’s secure classifiers (except for FORWARD).

Accuracy. MAXARGMAX introduces no numerical errors; the output corresponds exactly to the max and argmax of the values provides as inputs.

Runtime and Communication. A comprehensive evaluation of (arg)max protocols (together with significant bandwidth-optimizations) has already been given in Section 4.5.1 and we omit further measurements to avoid redundancy.

A.3 Evaluating Probability Mass Functions

EVALPROBMAS (Protocol 4.8, Section 4.4.4.1) is used in NAIVEBAYES, FORWARD, and VITERBI and we thus evaluate the primitive in these application contexts. In NAIVEBAYES, EVALPROBMAS is invoked to choose between the probability of $m$ possible feature values for each combination of the $k$ classes and $n$ features, which we batch into $1$-$m$-OT$_{kl}$. In FORWARD and VITERBI, EVALPROBMAS is invoked in a total of $T$ time steps for $N$ states and $M$ possible emissions, which we batch into $1$-$M$-OT$_{Tl}$. The overheads of EVALPROB are dominated by the those of the underlying Oblivious Transfer (OT) extension primitive [ALSZ13] and the reductions of $1$-$N$-OT to $1$-$2$-OT [NP01,NP05]. These steps are, in turn, dominated by the communication overheads of $mnl + \lceil\log_2(m)\rceil n(t + 2kl)$ bits for $1$-$m$-OT$_{kl}$ and $TMl + \lceil\log_2(M)\rceil T(t + 2Nl)$ bits for $1$-$M$-OT$_{Tl}$ with symmetric and asymmetric security parameters $t$ and $T$, respectively. Qualitatively, we thus expect runtime to grow linearly in all parameters.

Accuracy. EVALPROBMAS introduces no numerical errors beyond the fixed-point (logspace) number representation of its inputs (using $\mathcal{f2i}$ and $\mathcal{f2li}$).

Runtime and Communication. Figure A.1 plots the runtime of EVALPROBMAS for a representative parameter choice for FORWARD and VITERBI. We choose...
A.4. Logsum

Secure and accurate computation of logsum operations is challenging and incurs significant overheads in most prior works [PRT15, FDH+11, PRRS11, PRRS13]. Indeed, we have observed that LogSum (Protocol 4.11, Section 4.4.5) also dominates the runtime of our secure Forward protocol. Since secure computation with fixed-point precision in logspace is relevant beyond the scope of our work [FDH+10, CS10], e.g., for securely evaluating Gaussian Mixture Models [PRRS13], we provide a detailed evaluation of the LogSum primitive. The measured accuracy, runtime, and communication overheads for different choices of the number of approximation polynomials \( k \) and bitlength \( l \) are summarized in Table A.1.

### Table A.1: Evaluation of the secure LogSum primitive

<table>
<thead>
<tr>
<th>PLA size</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
<th>64</th>
<th>128</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bitlength ( l = 32 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Avg. abs. error</td>
<td>6.00e-2</td>
<td>4.40e-3</td>
<td>9.20e-4</td>
<td>5.30e-4</td>
<td>5.10e-4</td>
<td>5.40e-4</td>
<td>6.20e-4</td>
</tr>
<tr>
<td>Runtime (seq.)</td>
<td>2.35</td>
<td>2.83</td>
<td>3.30</td>
<td>4.22</td>
<td>5.79</td>
<td>9.23</td>
<td>14.90</td>
</tr>
<tr>
<td>Runtime (batch)</td>
<td>0.42</td>
<td>0.59</td>
<td>0.89</td>
<td>1.49</td>
<td>2.70</td>
<td>5.10</td>
<td>9.98</td>
</tr>
<tr>
<td>Communication</td>
<td>0.02</td>
<td>0.02</td>
<td>0.03</td>
<td>0.05</td>
<td>0.08</td>
<td>0.15</td>
<td>0.28</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PLA size</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
<th>64</th>
<th>128</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bitlength ( l = 64 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Avg. abs. error</td>
<td>6.00e-2</td>
<td>4.30e-3</td>
<td>7.70e-4</td>
<td>2.00e-4</td>
<td>5.50e-5</td>
<td>1.40e-5</td>
<td>2.70e-6</td>
</tr>
<tr>
<td>Runtime (seq.)</td>
<td>3.10</td>
<td>3.25</td>
<td>4.25</td>
<td>5.61</td>
<td>8.21</td>
<td>13.67</td>
<td>23.52</td>
</tr>
<tr>
<td>Runtime (batch)</td>
<td>0.75</td>
<td>1.00</td>
<td>1.52</td>
<td>2.61</td>
<td>4.66</td>
<td>8.88</td>
<td>17.74</td>
</tr>
<tr>
<td>Communication</td>
<td>0.03</td>
<td>0.04</td>
<td>0.05</td>
<td>0.08</td>
<td>0.14</td>
<td>0.26</td>
<td>0.49</td>
</tr>
</tbody>
</table>

We observe that the runtime of EvalProb grows linearly in \( T \), e.g., for fixed \( M = 10^4 \) increasing \( T \) from 10 to 100 also increases the runtime approximately by \( 10 \times \) from 0.04 s to 0.42 s. We also observe linear growth in \( M \), e.g., for fixed \( T = 100 \) increasing from 0.04 s for \( M = 10^3 \) to 0.42 s for \( M = 10^4 \). Finally, doubling the bitlength \( l \) also doubles the overheads, e.g., for fixed \( T = 100 \) and \( M = 10^4 \) increasing the bitlength from 32 bit to 64 bit increases runtimes from 0.42 s to 0.80 s.

The communication overhead (not shown in Figure A.1) qualitatively shows the same growth as runtime, e.g., requiring 0.06 MB for the smallest setup of \( (T = 10, M = 10^2, l = 32\text{ bit}) \) to 40.13 MB for the largest setup \( (T = 100, M = 10^4, l = 64\text{ bit}) \). This increase is linear in \( T, M, \) and \( l \) and confirms the theoretical complexity measure.

A fixed \( N = 10 \) (corresponding to the number of classes \( k \) in Naive Bayes) with bitlengths \( l = 32\text{ bit} \) (left) and \( l = 64\text{ bit} \) (right) varying the time steps \( T \) (corresponding to the number of features \( n \) in Naive Bayes) as well as the size of the emission alphabet \( M \) (corresponding to the number of possible feature values \( m \) in Naive Bayes). We increase \( M \) exponentially to show that EvalProbMass is efficient even for large probability mass functions.
Accuracy. LogSum uses a piece-wise linear approximation with $k$ intervals to compute the result. As expected, the average absolute error of the approximation decreases with increasing $k$. Increasing fixed-point precision by choosing $l = 64$ bit additionally improves accuracy, but only beyond $k = 8$. For the smaller values of $k$, the approximation error dominates the additional accuracy in the fixed-point representation. The maximum absolute error is always roughly one order of magnitude higher than the average error but shows the same decrease with regards to the piece-wise linear approximation (PLA) size $k$ and bitlength $l$. Increasing $k$ and $l$ comes at the cost of runtime and communication overheads as analyzed in the following.

Runtime and Communication. We first measure runtime for $m = 1000$ sequential LogSum operations and provide the average per single operation. The runtime grows linearly in $k$ and $l$ with $2.35 \text{ms}$ ($\sigma = 0.28 \text{ms}$) in our least accurate setup ($k = 2, l = 32$ bit) to $23.52 \text{ms}$ ($\sigma = 0.30 \text{ms}$) in the most accurate setup ($k = 128, l = 64$ bit). We now batch all $m$ operations into one single invocation of LogSum and again report the average runtime of a single logsum operation. Sequential operation requires evaluation of $m$ small Garbled Circuits (GCs) and requires $3m$ rounds of communication, whereas batched operation corresponds to the evaluation of one very large GC in only three rounds of communication. Thus, we expect batching to achieve the best speed-up for small circuits (small $k$ and $l$), where the sequential communication dominates the overheads. Indeed, we observe the best speed-up of $5.60 \times$ for $k = 2$ and $l = 32$ bit, which decreases, as expected, when increasing $k$ and $l$, e.g., down to $1.33 \times$ for the maximum $k = 128$ and $l = 64$ bit. We evaluated different batch sizes $m$ and observed no speedup beyond $m = 1000$. For both sequential and batched operation, we observe that to achieve a certain accuracy it is more efficient to increase the number of approximation intervals $k$ than the bitlength $l$.

Communication scales linearly in $k$ and $l$ and is in the order of tens to hundreds of kB per run of LogSum. The dominating part is the transmission of two $t$ bit keys per gate in the GC. Hence, communication overheads can be traded off against the security level $t$, e.g., switching to short-term security $t = 80$ reduces communication by approximately 34%. Ongoing research also considers reducing communication overheads of STC, e.g., through more efficient garbling schemes [ZRE15] or lookup tables [DKS+17].

A.5 Function Approximation

FunctionApprox (Protocol 4.6, Section 4.4.3.1) securely and efficiently approximates non-linear functions. Our main use case in SHIELD is the approximation of the logsum function as well as enabling arbitrary activation functions in ANN.

Accuracy. The accuracy of FunctionApprox strongly depends on the function that is approximated. The previous example of LogSum (which uses FunctionApprox to compute the function $f(x) = \log(1 + e^{-x})$) clearly shows that our approach achieves very accurate results when sufficiently many approximation intervals $k$ are used. We omit the accuracy evaluation of further examples since FunctionApprox (beyond contributing the core of LogSum) is currently only used in ANN which we evaluate against the approach by Dowlin et al. [DGL+16], which only uses linear activation functions.
Runtime and Communication. We note that the overheads of FunctionApprox when using polynomials of degree $d = 1$, i.e., a piece-wise linear approximation, are strictly bound by the overheads of the LogSum primitive that we just evaluated in detail. We also evaluated the overheads for polynomials of degree $d = 2$ and $d = 3$ and notice only slight increase of overheads, since the overheads of FunctionApprox on small degree polynomials are dominated by the constant overheads for the conversions to and from GCs and not by the computation of the approximation polynomial itself. For our most important application of FunctionApprox, i.e., the approximation of the logsum function, increasing the number of approximation intervals turned out to be more effective w.r.t. performance and accuracy than increasing the degree of the approximation polynomials and thus omitted further evaluation.
<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES</td>
<td>Advanced Encryption Standard</td>
</tr>
<tr>
<td>ANN</td>
<td>Artificial Neural Network</td>
</tr>
<tr>
<td>BGW</td>
<td>Ben-Or-Goldwasser-Wigderson</td>
</tr>
<tr>
<td>DGK</td>
<td>Damgård-Geisler-Kreigaard</td>
</tr>
<tr>
<td>DKG</td>
<td>Distributed Key Generation</td>
</tr>
<tr>
<td>DoS</td>
<td>Denial of Service</td>
</tr>
<tr>
<td>ECDSA</td>
<td>Elliptic Curve Digital Signature Algorithm</td>
</tr>
<tr>
<td>ECIES</td>
<td>Elliptic Curve Integrated Encryption Scheme</td>
</tr>
<tr>
<td>FHE</td>
<td>Fully Homomorphic Encryption</td>
</tr>
<tr>
<td>GC</td>
<td>Garbled Circuit</td>
</tr>
<tr>
<td>GMW</td>
<td>Goldreich-Micali-Wigderson</td>
</tr>
<tr>
<td>GPS</td>
<td>Global Positioning System</td>
</tr>
<tr>
<td>HE</td>
<td>Homomorphic Encryption</td>
</tr>
<tr>
<td>HMM</td>
<td>Hidden Markov Model</td>
</tr>
<tr>
<td>LBS</td>
<td>location-based service</td>
</tr>
<tr>
<td>LPPM</td>
<td>location privacy protection mechanism</td>
</tr>
<tr>
<td>LSS</td>
<td>Linear Secret Sharing</td>
</tr>
<tr>
<td>NIST</td>
<td>National Institute of Standards and Technology</td>
</tr>
<tr>
<td>OPSI</td>
<td>Outourced Private Set Intersection</td>
</tr>
<tr>
<td>OSM</td>
<td>OpenStreetMap</td>
</tr>
<tr>
<td>OT</td>
<td>Oblivious Transfer</td>
</tr>
<tr>
<td>PETs</td>
<td>Privacy Enhancing Technologies</td>
</tr>
<tr>
<td>PFE</td>
<td>Private Function Evaluation</td>
</tr>
<tr>
<td>PHE</td>
<td>Partially Homomorphic Encryption</td>
</tr>
<tr>
<td>PII</td>
<td>personally identifiable information</td>
</tr>
<tr>
<td>PKC</td>
<td>Public Key Cryptography</td>
</tr>
<tr>
<td>PLA</td>
<td>piece-wise linear approximation</td>
</tr>
<tr>
<td>PRML</td>
<td>pattern recognition and machine learning</td>
</tr>
<tr>
<td>PRNG</td>
<td>pseudorandom number generator</td>
</tr>
<tr>
<td>PSI</td>
<td>Private Set Intersection</td>
</tr>
<tr>
<td>RMSS</td>
<td>Random Modular Subset Sum</td>
</tr>
<tr>
<td>RSA</td>
<td>Rivest-Shamir-Adleman</td>
</tr>
<tr>
<td>RSSI</td>
<td>received signal strength indicator</td>
</tr>
<tr>
<td>RTT</td>
<td>Round Trip Time</td>
</tr>
<tr>
<td>SFE</td>
<td>Secure Function Evaluation</td>
</tr>
<tr>
<td>SIMD</td>
<td>Single-Instruction-Multiple-Data</td>
</tr>
<tr>
<td>SMC</td>
<td>Secure Multiparty Computation</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>SNP</td>
<td>Single Nucleotide Polymorphism</td>
</tr>
<tr>
<td>STC</td>
<td>Secure Two-Party Computation</td>
</tr>
<tr>
<td>SVM</td>
<td>Support Vector Machine</td>
</tr>
<tr>
<td>SWHE</td>
<td>Somewhat Homomorphic Encryption</td>
</tr>
<tr>
<td>TLS</td>
<td>Transport Layer Security</td>
</tr>
<tr>
<td>TTP</td>
<td>Trusted Third Party</td>
</tr>
<tr>
<td>VSS</td>
<td>Verifiable Secret Sharing</td>
</tr>
<tr>
<td>ZKP</td>
<td>Zero-Knowledge Proof</td>
</tr>
<tr>
<td>iDASH</td>
<td>Integrating Data for Analysis, Anonymization, and Sharing</td>
</tr>
<tr>
<td>OPSI-CA</td>
<td>Outsourced Private Set Intersection Cardinality</td>
</tr>
<tr>
<td>PSI-CA</td>
<td>Private Set Intersection Cardinality</td>
</tr>
</tbody>
</table>
Bibliography


[SAS13] Latanya Sweeney, Akua Abu, and Julia Winn. Identifying Participants in the Personal Genome Project by Name. Available at SSRN 2257732, 2013.


## Curriculum Vitae

### Personal Details

<table>
<thead>
<tr>
<th>Last Name</th>
<th>Ziegeldorf</th>
</tr>
</thead>
<tbody>
<tr>
<td>First Name</td>
<td>Jan Henrik</td>
</tr>
<tr>
<td>Date of Birth</td>
<td>June 5, 1986</td>
</tr>
<tr>
<td>Place of Birth</td>
<td>Hagen, NRW, Germany</td>
</tr>
<tr>
<td>Nationality</td>
<td>German</td>
</tr>
</tbody>
</table>

### Education

<table>
<thead>
<tr>
<th>High School</th>
<th>Ruhrtal-Gymnasium, Schwerte</th>
</tr>
</thead>
<tbody>
<tr>
<td>1996 – 2005</td>
<td>Abitur: June 2005</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>University</th>
<th>RWTH Aachen University</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Degree: Dipl.-Inform.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PhD Student</th>
<th>RWTH Aachen University</th>
</tr>
</thead>
<tbody>
<tr>
<td>2012 – 2017</td>
<td>Chair of Communication and Distributed Systems</td>
</tr>
<tr>
<td></td>
<td>Adviser: Prof. Dr.-Ing. Klaus Wehrle</td>
</tr>
</tbody>
</table>