Network Management in a World of Secrets

Wilko Henecka

Thesis submitted for the degree of

Doctor of Philosophy

in

Applied Mathematics

at

The University of Adelaide

(Faculty of Mathematical and Computer Sciences)

Department of Applied Mathematics

May 5, 2015

Contents

List of Tables

List of Figures

Signed Statement

This work contains no material which has been accepted for the award of any other degree or diploma in any university or other tertiary institution and, to the best of my knowledge and belief, contains no material previously published or written by another person, except where due reference has been made in the text.

I consent to this copy of my thesis, when deposited in the University Library, being available for loan and photocopying.

SIGNED: . . . . . DATE: . . . . . .

Acknowledgements

I would like to thank my supervisors Matthew Roughan and Nigel Bean; the funding institutions for my scholarships (the University of Adelaide and the Defence Science Innovation Centre); paper co-author Thomas Schneider; and lastly both my partner Hanna, and daughter Pipaluk, for their contributions to this thesis. Without you the following pages would still be trees.

Summary

The Internet is by definition a network of networks. No-one operates, or controls more than a small slice of the overall Internet. A large number of network providers manage their own, independent networks, and these then interconnect to create the world-spanning artefact we call the Internet. These Internet Service Providers (ISPs) are highly heterogeneous – they vary widely in scale, technology, objective, and management policies. However, they have some general features in common, and in particular they all engage in some kind of network management.

Network management consists of a variety of tasks and activities, including performance analysis, detection of network anomalies or fraud, traffic engineering, and capacity planning.

Some of the activities we wish to conduct as part of these tasks are apparently not possible without co-operation. However, these companies also compete, and much of the information that they would need to share to co-operate is secret!

For the last 40 years a field of cryptology has been growing, named *Secure Multiparty Computation* (SMC). It offers protocols for multiple parties to compute functions without revealing their respective inputs to each other. These techniques have come a long way from the earliest theoretical ideas, now offering practical solutions for wide-ranging problems including electronic voting, auctions, or data mining. Despite the maturity of this exciting work on SMC, there remains relatively little research in the context of network management.

In this thesis, we apply concepts of SMC to problems within network management.

Importantly, SMC techniques generate significant overheads because they use cryptographic primitives to enable secure computation. With this in mind, the first half of this thesis we attempt to improve the practicability of SMC. Specifically, we show a new implementation of Yao's two-party secure function evaluation protocol with significantly better performance than previous implementations. Its low memory footprint enables the evaluation of bigger circuits with less memory.

We then demonstrate a secure scaling protocol which enables two parties to convert real-numbered privacy-preserving problems into the integer domain by scaling, as almost all techniques for SMC support only integer inputs and operations. Our approach does not limit the choice of SMC techniques for the privacy-preserving problem, nor does it introduce additional overheads compared with previous solutions. The core of our protocol is a novel algorithm for privacy-preserving random number generation.

In the second half of the thesis, we present two examples of how to apply SMC to network management.

First we present a distance-vector routing protocol that allows routers to compute the shortest paths without learning the distances of any paths. Whereas previous solutions relied on trusted third parties for route computation, we maintain the distributed nature of a routing protocol. The basic components of the protocol can easily be extended to implement other types of path metrics.

The second example is a protocol for privacy-preserving fraud detection. It enables telecommunication providers to co-operate in detecting subscription fraud without violating protection laws for phone records. We present several protocols for call-signature based subscription fraud detection with different levels of privacy, enabling providers to mine each others fraudster databases.

In all cases we present implementations, and we use these to show that SMC-based protocols are practical and may be highly beneficial to network operators.

Chapter 1

Introduction

The Internet, which started as a single research network, soon grew to be a large collection of interconnected autonomous networks. With the opening of the Internet to commerce in the late 1980s companies started to appear which provided Internet access to businesses and home users. This development changed the Internet from having a research and education focus to the diverse multipurpose network of today. These Internet Service Providers (ISPs) are highly heterogeneous - they vary in scale, technology, objective and management policies. However, they all have to engage in some kind of network management. It consists of a variety of tasks.

- *Routing* involves exchanging reachability information and finding routes through the Internet to all destinations to provide access to the Internet.

- *Traffic Engineering* is the optimisation of routing policies according to traffic management goals. We distinguish between intra- and inter-domain traffic engineering; that is, optimising routing policies within one network or optimising between several networks.

- *Performance Analysis* is a part of quality control. To maintain and improve the quality of service, measurements are essential.

- *Capacity Planning* is the optimisation of link capacity and choice of location.

- *Detection of network anomalies* is essential in network management in order to react quickly to outages, attacks and fraudulent customers.

Some of these tasks require co-operation with other networks, for instance, routing cannot work if there is no exchange of reachability information. Other tasks would strongly benefit from co-operation. For example, co-operative anomaly detection can locate the anomaly more easily and precisely, and co-ordinate better counter measures. Selfish inter-domain traffic engineering falls short in performance compared with co-operative traffic engineering [99]. Internet performance is an issue of great interest, but it is not trivial to measure across multiple networks without co-operation.

Pertinently, although co-operation may be beneficial for the ISPs, they also compete and much of the information they would need to share to co-operate is considered secret.

1.1 Secrets in the World of Networks

There are several kinds of information an ISP may not want to reveal to its competitors.

- Topology: the physical and logical structure of the network.

- Policies: a description of the business relationships between neighbouring ISPs, the configuration of the network, and the different QoS for different customers.

- Operational Data: for example, the utilisation of the network.

- Customer information: names, locations, usage history and current contracts.

The reasons for keeping these kinds of information secret can be:

- Commercial concerns: the network topology or policies might give the ISP an advantage over its competitors.
- Privacy: an ISP might be obligated to ensure its customers' privacy either through legislation or customer agreements.
- Embarrassment / fear of the unknown: the relationship of trust between the ISP and its customers, and its overall image would suffer in the case of a disclosure of poor network design or management. The description of 'poor' might be attached ex post facto to a network management technique – hence fear of unknown future consequences.
- Security: attackers may use information about the topology of a network to fine-tune their attacks.

1.2 Information Sharing

As there is no centralised authority managing the Internet, the ISPs have to co-operate to make it work. But co-operation is not possible without information sharing. We delineate between two such types:

Mandatory Sharing In order to make the Internet functional, the ISPs must share certain information. For instance, routing could not take place if the ISPs did not share reachability information.

Voluntary Sharing As explained earlier, co-operation can be beneficial for ISPs. They may gain an advantage if they share certain information voluntarily.

ISPs perform mandatory sharing, but only perform voluntary sharing in isolated cases. The conflict between privacy and sharing is leaning towards

privacy. It appears that the anticipated losses through sharing seem to be greater than the expected gains. In this thesis we present techniques to enable co-operation without the need to reveal private information, thus making co-operation more attractive.

1.3 Secure Multiparty Computation

Secure Multiparty Computation (SMC) protocols enable parties to carry out distributed computation tasks without revealing their inputs to one another. At the end of the protocol no party knows anything except its own input and the result of the computation.

Example: Secure Sum Alice, Bob, Charlie and Dora want to know their average wage. However, they do not want to reveal their respective wages to each other. One solution for this problem is to engage in a secure sum protocol, as follows:

- First they have to agree on an Integer n which must be bigger than the possible sum of their wages, and which defines the ring $\mathbb{Z}_n$ over which all arithmetic will be done.
- Alice picks an element r uniformly at random out of $\mathbb{Z}_n$. She then adds her wage w_A and sends $x_A = r + w_A$ to Bob.
- Bob then adds his wage w_B and sends $x_B = x_A + w_B$ to Charlie.
- Charlie and Dora act similarly to Bob, in sequence.
- Finally, Alice will receive $x_D = x_C + w_D$ from Dora. Alice can now compute the sum of all wages by subtracting r from x_D. Dividing the sum by four will give the average wage.

The reason why no party can learn another parties' input is that each party receives exactly one message containing one value x_i. The value of x_i can

be any element of $\mathbb{Z}_n$ with the same probability. Thus it is impossible to distinguish between an x_i and a randomly generated value containing no information about the (partial) sum.

However, if we allow some parties to collude, then they may readily compute another parties input. For example, Bob and Dora could compute Charlie's input by subtracting x_B from x_C. For a secure sum protocol which is secure against such colluding parties, see [135] and the citations therein.

As such, it is important to specify the context in which the security holds. This context is called the *security model*. Within a security model, it is specified what assumptions are made, plus how powerful an attacker may be. Generally, protecting against powerful attackers is more expensive than protecting agains weaker attackers, and thus there is a trade-off between protection and performance.

Secure sum is a special purpose protocol, given that it only allows for computing a sum. There are general purpose protocols which can compute any functionality. They can be based on different cryptographic primitives, such as secret sharing, oblivious transfer, homomorphic encryption or garbled circuits. In this thesis we focus on the latter: namely homomorphic encryption and garbled circuits, which are described in detail in Chapter 2.

1.3.1 SMC in the Network World

To date, the major use of applying privacy-preserving computation techniques to problems in the network world has centred on the field of measurements. Although each AS in the Internet has a variety of different measurements of its network, the efforts of aggregating these may be hampered by the fear of sharing such (potentially sensitive) information. In 2006, Roughan and Zhang published various solutions in this regard, including secure summation of Internet traffic, distributed network problem detection [131] and aggregation of performance measurements [130]. More recently, Burkhart *et al.* [29] presented SEPIA, a ready-to-use library for privacy-preserving

aggregation of network data. Ricciato and Burkhart [126] further improved the efficiency of collaborative measurements by both relaxing the security constraints for intermediate results, and by separating the computation into an offline phase for pre-computations, plus an online phase.

Furthermore, Brickell and Shmatikov [26] demonstrated a solution for the All-Pairs-Shortest-Distance and Single-Source-Shortest-Distance problem for two parties on their joint graph. This result might be used as a building block for traffic engineering protocols. In contrast, Roughan and Zhang used a different approach in their inter-domain traffic engineering protocol GATEway [132], by choosing a genetic algorithm to find the optimal routing solution.

A special case of SMC is anonymisation. In an abstract way your identity can be an input to an algorithm as well. An illustrative example of anonymisation in the network world is the 'Tor project', presented in 2004 by Dingledine *et al.* [47]. It is a service distributed across several servers that allows users to participate on the Internet without revealing their identity.

1.4 Thesis Roadmap

Beyond this introduction, this thesis is split into six chapters. Chapter 2 provides a background on secure multiparty computation, focusing on homomorphic encryption and garbled circuits. (Please note that in each following chapter we provide the background required for the specific issue being addressed).

The first half of our research contribution focuses on the practicability of SMC. In Chapter 3, we present a new implementation of Yao's garbled circuit based two-party secure function evaluation protocol. Specifically, we present several optimisations that result in lower memory consumption and significantly better performance compared to previous implementations.

In Chapter 4, we propose a secure scaling protocol that allows two par-

ties to map their real number inputs into integers without revealing any information about their respective inputs, as almost all techniques for SMC support only integer inputs and operations. The main component is a novel algorithm for privacy-preserving random number generation.

In Chapters 5 and 6 we apply SMC techniques to problems of network management. We present a privacy-preserving routing protocol in Chapter 5 with the aim of protecting the privacy of the routing configuration. In Chapter 6, we propose several protocols for detecting subscription fraud in telecommunication networks. They are based on different SMC techniques, and offer correspondingly different levels of privacy and performance. We show feasibility and provide a comparison with implementations of our protocols.

We conclude the thesis in Chapter 7.

1.5 Statement of Research Contributions

The research contributions of this thesis have previously been published. Each of the Chapters 3, 4, 5, and 6 map to one of the papers listed below. I have been the principal author of all but the paper regarding faster secure two-party computation, where both authors contributed equal parts.

1.5.1 Publications Arising From This Thesis

Components of this thesis have previously been published:

- *Conversion of Real-Numbered Privacy-Preserving Problems into the Integer Domain*, Wilko Henecka, Nigel Bean, and Matthew Roughan. In Proceedings of the 14th International Conference on Information and Communications Security (ICICS), October 2012, Hong Kong, LNCS 7618 pp.131–141.

- *Faster Secure Two-Party Computation with Less Memory*, Wilko He-

necka and Thomas Schneider. In Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security (ASIA CCS), May 2013, Hangzhou, pp. 437–446.

- *STRIP: Privacy-Preserving Vector-Based Routing*, Wilko Henecka and Matthew Roughan, In Proceedings of the 21st IEEE International Conference on Network Protocols (ICNP), October 2013, Göttigen.

- *Privacy Preserving Fraud Detection Across Multiple Phone Record Databases*, Wilko Henecka and Matthew Roughan, Accepted on the 1st of December 2014 to appear in IEEE Transactions on Dependable and Secure Computing, DOI: 10.1109/TDSC.2014.2382573.

Chapter 2

Background

This chapter provides the background to the secure multiparty computation techniques we use throughout this thesis. Additional background that is specific to certain chapters is provided in the respective chapters.

2.1 Secure Multiparty Computation

Secure multiparty computation protocols are a set of techniques, often cryptographic in nature, which enable parties to carry out distributed computation tasks without having to reveal their private data.

Perhaps the most famous problem in the area is called the Millionaire's Problem, where two millionaires meet on the street, and wish to determine who is wealthier, but without revealing their own wealth. It was shown previously by Yao [144] that any polynomial time function can be computed in a secure distributed manner, thus providing a simple solution to the Millionaire's Problem.

That said, Yao's approach [144] may not always be the most economical choice, though there is now a substantial literature on secure multiparty computation, and the closely related areas of privacy-preserving data mining and private information retrieval.

A private information retrieval protocol allows a user to query an item

from a database, held by another party, without revealing which item is retrieved (see *e.g.* [34, 59]). In contrast, privacy-preserving data mining protects the confidentiality of the items in a database whilst providing means to extract "meaningful" information of aggregated items (see *e.g.* [4, 96]).

Both settings have in common that the information flow is asymmetric. One or some parties hold private inputs, whilst other parties learn the output of a particular function on these inputs. In this thesis, we are interested in protocols where the confidential information is somewhat equally spread amongst all cooperating parties.

The security of a secure multiparty computation protocol is defined by a *security model*. The security model describes how powerful an adversary is: that is, what he can and cannot do. Some parameters in defining a security model include set-up assumptions, the communication channels, computational limitations, restricted adversarial behaviour, restricted notions of security, and upper bounds on the number of dishonest parties (see [64] for a detailed explanation).

Intuitively, a more powerful adversary requires more sophisticated protection mechanisms, costing more resources. Choosing the right protocol for a specific use-case therefore becomes a trade-off between security level and efficiency.

In the last few decades SMC has come a long way from its first feasibility results to actual implementations in the real world. The first deployment of SMC was in Denmark in 2008, where it was used in a secure double auction to determine the price of sugar beets [19]. In 2011, ITL, an Estonian non-governmental non-profit organisation with the goal of promoting cooperation between ICT companies, deployed an application for collecting and reporting of financial indicators of the ICT sector [18].

2.2 Homomorphic Encryption

An encryption scheme is called *homomorphic* if there exists an operation on two ciphertexts that is equivalent to another operation on the corresponding plaintexts: *i.e.*,

$$\mathrm{Enc}(x) \odot \mathrm{Enc}(y) = \mathrm{Enc}(x \oplus y),$$

for some operations $\odot$ and $\oplus$, with $\mathrm{Enc}(x)$ denotes the encryption of x.

An encryption scheme consists of three algorithms *(Gen, Enc, Dec)* for key generation, encryption, and decryption. There are two categories of encryption schemes: secret-key (or symmetric) encryption and public-key (or asymmetric) encryption. Whereas in symmetric encryption the same key is used both for encryption and decryption, in asymmetric encryption there are two different keys: specifically, the public key is published and used for encrypting, while the private key is kept private and used for decrypting. This approach eliminates the need for the parties to establish a shared secret before exchanging encrypted messages. This key distinction between secret-key and public-key encryption schemes is crucial for building protocols for privacy-preserving co-operation.

In secret-key encryption every party that knows the key can decrypt. To be able to add your own input to the protocol you need to know the key, however, this would also enable you to decrypt the other parties inputs.

Therefore we will only consider homomorphic public-key encryption in this thesis. (Please note that there are other scenarios where homomorphic secret-key encryption can be used, for instance, private data processing in the cloud [33, 134]).

We classify a scheme as *additively homomorphic* if the operation $\oplus$ corresponds to standard arithmetic addition, and *multiplicatively homomorphic* if the operation $\oplus$ corresponds to standard arithmetic multiplication.

Examples of multiplicatively homomorphic encryption schemes are RSA [127] and ElGamal [50].

The first additively homomorphic encryption system was published by Goldwasser and Micali in 1982 [65]. It allows for inputs with a size of only 1 bit, and has a rather large expansion ratio - that is, a single bit of plaintext is encrypted as an integer modulo n, with n being a product of two large primes. However, it laid the basis for subsequent encryption systems. Benaloh [35] proposed a generalisation that allows for longer blocks of data to be encrypted at once. Naccache and Stern [106] improved upon Benaloh's scheme, as their system allows for bigger blocks of data coexistent with a smaller expansion ratio. With a change of the base group, Okamoto and Uchiyama [112] achieved an expansion ratio of 3. Paillier's scheme [114] is an improvement of the previous scheme. It has an expansion ratio of 2, yet comparatively very efficient encryption and decryption functions. As such, it is the most widely used additively homomorphic encryption system today. As we make subsequent use of it in this thesis, it will be explained in more detail in an upcoming section.

Damgård and Jurik [44] proposed a generalisation of Paillier's scheme. It uses groups of the form $\mathbb{Z}^*_{n^{s+1}}$ with Paillier's scheme being the special case for $s = 1$. The expansion ratio is $1 + 1/s$. With increasing s the computational complexity also increases. This means one can "buy" better communication complexity for the cost of worse computation complexity. Since communication costs in our intended scenarios are not critical, we will use Paillier's homomorphic encryption scheme, as it offers the best computational complexity.

The homomorphic property makes all such encryption schemes useful tools for privacy-preserving computation, as they allow combinations of messages in the encrypted space, (*i.e.*, the values of those messages are kept private).

Notwithstanding, however, a homomorphic encryption scheme can only be useful for privacy-preserving computation if it is *semantically secure*.

2.2.1 Semantic Security

A public-key cryptosystem is considered semantically secure if it is infeasible for a computationally-bounded adversary to derive information about the plaintext -specifically if given only the corresponding ciphertext and the public encryption key.

As semantic security is equivalent to indistinguishability against chosen ciphertext attacks (IND-CPA) [141] we will give the definition for the latter.

Definition 1. *IND-CPA is defined by the following game between an adversary and a challenger:*

1. *The challenger generates a key pair and publishes the public key to the adversary.*
2. *The adversary choses two plaintexts and sends them to the challenger.*
3. *The challenger selects one of the messages by flipping a fair coin, encrypts it with the public key, and sends the ciphertext to the adversary.*
4. *The adversary is free to perform additional polynomial time computations. Finally, it outputs a guess which of the two messages was chosen by the challenger.*

A cryptosystem is indistinguishable under a chosen plaintext attack if every attacker, modelled by a probabilistic polynomial time Turing machine, wins the game with only negligible advantage over random guessing.

It is easy to see that (textbook) RSA is not semantically secure, as its encryption function is deterministic. That is, a plaintext will always be encrypted to the same ciphertext. The adversary can therefore always win the game by encrypting the two messages himself and comparing the ciphertexts to the challenge.

As such, real-world implementations of RSA use an additional padding scheme (for example, OAEP [10]) to randomise encryption. However, with the use of padding, RSA loses the homomorphic property.

It is worth noting that all of the other homomorphic cryptosystems listed above are semantically secure.

2.2.2 Paillier's Encryption Scheme

Paillier's encryption scheme [114] is the most widely used additively homomorphic cryptosystem. It provides comparatively efficient algorithms for encryption and decryption, and has a expansion ratio of 2 from the plaintext to the ciphertext space. Its security is based on the *Decisional Composite Residuosity Assumption* (see also definition in [114]), that is, given a composite n and an integer z, there exists no polynomial time algorithm to decide whether z is a n-th residue modulo n^2 or not.

The scheme consists of three algorithms for key generation, encryption and decryption, respectively.

Key generation Let k be the security parameter. Choose two k-bit primes p and q uniformly at random and set $n = pq$. Large primes can be generated efficiently by first generating random numbers of appropriate size and then testing them for primality using efficient algorithms like Miller-Rabin [87, page 394] (see also [3, Appendix A] for a detailed description). Select a random $g \in \mathbb{Z}^*_{n^2}$ and ensure that n divides the order of g by checking $\gcd(L(g^\lambda \mod n^2), n) = 1$ with $L(u) = \frac{u-1}{n}$ and $\lambda = \mathrm{lcm}(p-1, q-1)$.

The public key is (n, g) and the private key is (λ).

Encryption To encrypt a message $m < n$, one chooses a random integer $r \in \mathbb{Z}^*_n$ and computes the ciphertext as

$$c = g^m r^n \mod n^2.$$

Decryption Let $c \in \mathbb{Z}^*_{n^2}$ be the ciphertext to decrypt. The corresponding plaintext message is computed as

$$m = \frac{L(c^\lambda \mod n^2)}{L(g^\lambda \mod n^2)} \mod n.$$

Properties

- Paillier's scheme is additively homomorphic, because

$$\begin{aligned}\text{Enc}(m_1)\,\text{Enc}(m_2) &= g^{m_1} r_1^n \cdot g^{m_2} r_2^n \mod n^2 \\ &= g^{m_1+m_2}(r_1 r_2)^n \mod n^2 \\ &= \text{Enc}(m_1 + m_2).\end{aligned}$$

- Homomorphic multiplication with a constant k:

$$\begin{aligned}\text{Enc}(m)^k &= (g^m r^n)^k \mod n^2 \\ &= g^{km}(r^k)^n \mod n^2 \\ &= \text{Enc}(km).\end{aligned}$$

- Self-blinding - one can change the ciphertext without changing the value of the original plaintext:

$$\begin{aligned}\text{Enc}(m) \cdot g^{nx} &= g^m r^n \cdot g^{nx} \mod n^2 \\ &= g^{m+nx} r^n \mod n^2 \\ &= \text{Enc}(m), \\ \text{as } m + nx &= m \mod n.\end{aligned}$$

- Paillier's scheme is semantically secure. Intuitively, the random component r^n of an encryption ensures that it is very unlikely to encrypt a plaintext to the same ciphertext. Consequently, this makes it impossible to distinguish between two ciphertexts. See [114, Theorem 15] for a formal proof.

Efficiency Considerations

- Damgård and Jurik [44] proposed to choose $g = (1 + n)$. This leads to a simplified encryption function:

$$\begin{aligned}c &= (1 + n)^m r^n \mod n^2 \\ &= (1 + nm) r^n \mod n^2.\end{aligned}$$

The determing factor for the computational complexity of an encryption is the number of exponentiations. By choosing $g = 1 + n$ we can replace one expensive exponentiation with a cheap multiplication, almost halving the computational complexity.

- As Paillier [114] pointed out, the decryption function can be computed more efficiently by applying the Chinese Remainder Theorem (CRT) [46]. Let

$$L_p(u) = \frac{u-1}{p} \text{ and } L_q(u) = \frac{u-1}{q}.$$

Decryption can then be made faster by first computing the message mod p and mod q and then combining the two using the CRT‘:

$$\begin{aligned} m_p &= \frac{L_p(c^{p-1} \mod p^2)}{L_p(g^{p-1} \mod p^2)} \mod p \\ m_q &= \frac{L_q(c^{q-1} \mod q^2)}{L_q(g^{q-1} \mod q^2)} \mod q \\ m &= \mathrm{CRT}(m_p, m_q) \mod pq \end{aligned}$$

We now have to perform more than twice the number of computations, yet the moduli in these computations have only half the bit size of n, which leads to a factor of 4 efficiency improvement.

- The denominator in the decryption function is always the same for a given private key (it depends only on g, p and q), therefore, one can precompute the denominator in the key generation phase and store the result in the private key.

2.2.3 Fully Homomorphic Encryption

Fully homomorphic encryption systems allow for both addition and multiplication under encryption.

The almost fully homomorphic schemes of Boneh *et al.* [22] and Gentry *et al.* [63] allow for a polynomial number of additions and one multiplication.

The *first* fully homomorphic encryption schemes were proposed by Gentry in 2009 [60, 61]. However, the computational complexity and ciphertext sizes for reasonable security levels are impractical.

Although significant efforts from the research community over the past few years have been invested into improving Gentry's ideas, the practicality of fully homomorphic encryption has remained significantly limited.

For example, Gentry *et al.* [62] showed an implementation of an evaluation of the AES circuit. It needed several hours to compute one AES encryption, compared with just milliseconds for the garbled-circuit approach (discussed in Section 3.5).

Intuitively, it seems unlikely that fully homomorphic encryption will reach the efficiency of current public-key cryptosystems. For instance, they require the addition of extra algebraic structure to allow for the homomorphic operations. But as these structures weaken security, additional countermeasures, with extra costs, are necessary.

Still, homomorphic encryption schemes, especially the simple additive ones, can be put to good use.

2.3 Garbled Circuits

Another method for efficient privacy-preserving computation is based on *Garbled Circuits* (GC). The idea of GC is rooted in Yao's work regarding secure function evaluation (SFE) [143, 144]. The key concept is to represent the function f to be evaluated as a circuit. GC techniques provide means to enable one party, given the inputs in encrypted form, to evaluate the circuit without learning any meaningful information about the intermediate values.

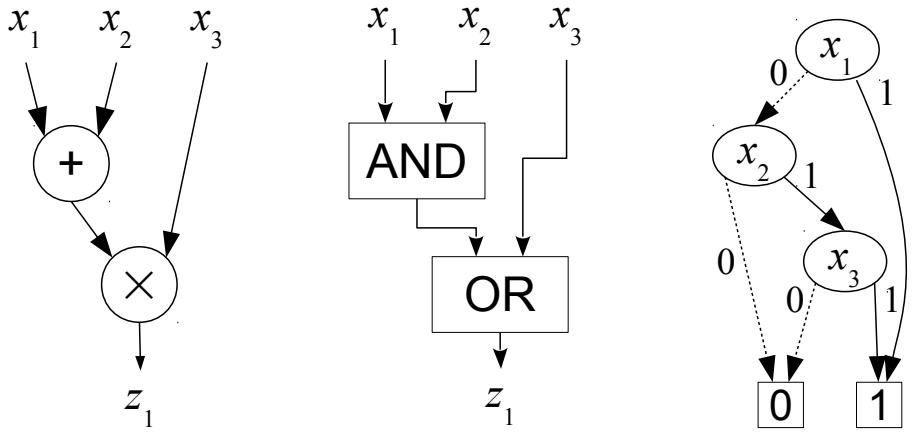
2.3.1 Function Representation as a Circuit

A circuit is a directed acyclic graph. Nodes have k inputs and one output and represent either an atomic operation or an input to the circuit. Edges

connect the output of one node to inputs of other nodes and pass intermediate values from one node to the others.

Choosing an appropriate set of atomic operations, every function can be expressed as a circuit. As a node can only compute the value of its output once all inputs are available, sorting the nodes in a topological order ensures a seamless sequential evaluation of the circuit.

There are three types of circuits used for SFE:



(a) Arithmetic circuit for $z_1 = (x_1 + x_2)x_3$

(b) Boolean circuit for $z_1 = (x_1 \wedge x_2) \vee x_3$

(c) OBDD for $x_1 \vee (x_2 \wedge x_3)$

Figure 2.3.1: Function representations as circuits

Arithmetic Circuits An arithmetic circuit (*e.g.* Figure 2.3.1a) is defined over a ring R, *e.g.*, $\mathbb{Z}_p$. The atomic operations are $+$ and $\times$, denoting addition and multiplication in R. All values passed through edges are elements in R.

Boolean Circuits Let F be the set of atomic Boolean functions allowable in the circuit model. A Boolean circuit over set F with n inputs and m outputs is then defined as a finite directed acyclic graph. Each node corresponds to either a basis function or an input and has d inputs and one output. Edges connect outputs of one node to inputs of other nodes.

A typical set of atomic functions in the context of privacy-preserving computing contains the AND, OR, NOT and XOR functions.

Again, the nodes can be sorted in a topological order, such that the i-th node has no inputs that are outputs of a successive node.

Although every function can be expressed as a Boolean circuit with an adequate set of atomic functions, finding the representation with the least amount of nodes is "very hard" [84] (it is unlikely to be solvable in polynomial time).

Boolean circuits can be generated automatically from a higher-level specification of the function. The Fairplay compiler [101] uses the Secure Function Description Language (SFDL) as the input language to describe the function. It supports variables, functions, Boolean ($\wedge, \vee, \oplus, \ldots$), arithmetic ($+, -, \times, \ldots$) and comparison ($<, >, =, \ldots$) operators and some control structures. The compiler automatically transforms a function described in SFDL into a corresponding Boolean circuit.

TASTY [71], the Tool for Automating Secure Two-partY computations, provides the input language TASTYL, a subset of the Python language, to describe the function. It not only allows for generation of garbled circuits but also for secure computation using homomorphic encryption.

Another circuit generator was proposed by Huang *et al.* [78]. Using an object-oriented representation for the function, they provide higher-level objects, which describe operators such as addition of 2 l-bit values. It is designed to be easily extensible to implement any operation. However, this representation does not directly support control structures.

Ordered Binary Decision Diagrams (OBDD) [27] Another way of representing a Boolean function is an ordered binary decision diagram. A binary decision diagram (BDD) can represent any Boolean function of the form $f : B^k \to B$, where $B = \{0, 1\}$, as a rooted, directed acyclic graph; this consists of both decision nodes and two terminal nodes (called the 0-

terminal and the 1-terminal). Each decision node i is labeled by Boolean input variable x_i and has two child nodes, named the low and high child. The edge from a node to a low (high) child represents an assignment of the variable to 0 (1). A BDD is called *ordered* if the variables appear in the same order on all paths from the root to the terminals.

This technique can be extended to describe Boolean functions with outputs bigger than one bit, *i.e.*, $f : B^k \to B^l$, by using multiple OBDDs, where the i-th OBDD computes the i-th output bit.

The size of the OBDD is determined both by the function being represented and the chosen ordering of the variables. The problem of finding the variable ordering which leads to the smallest circuit is NP-hard [20].

2.3.2 Secure Function Evaluation of Circuits

A secure function evaluation (SFE) protocol has to provide means which allow the participating parties to evaluate a circuit in such a way that no party can learn the other parties' input, nor any intermediate value of the circuit computation, as they might reveal information about the inputs.

Arithmetic Circuits As the two atomic operations in an arithmetic circuit are $+$ and $\times$ the obvious cryptographic tool of choice is fully homomorphic encryption. A two-party SFE protocol is then very simple:

- The client encrypts his inputs and sends these encrypted values, together with the corresponding public key, to the server.
- The server computes the function as described in the arithmetic circuit by using the fully homomorphic property of the cryptosystem, and returns the result to the client, who then decrypts it.

It is easy to see that neither party can learn anything about the other party's input, as long as the security of the cryptosystem holds.

However, as fully homomorphic encryption is still far from practical, other strategies to evaluate arithmetic circuits [89] are more efficient.

2.3.3 Garbled Circuits: SFE of Boolean Circuits and OBDD's

In this thesis we focus on SFE of Boolean circuits, as research on this technique is more advanced. SFE of OBDD's can either be performed analogously to SFE of Boolean Circuits (see *e.g.*, [92]), or via use of a protocol based on homomorphic encryption, with better communication efficiency yet higher computation costs [80].

The essence of garbled circuits is to represent the function f as a Boolean circuit, and to subsequently associate two (random) tokens to each edge of the circuit; notably the tokens have the hidden semantics of 0 and 1 (the true value of the edge is *garbled*). By providing means to propagate these tokens across the nodes whilst maintaining the hidden semantics, one can create a two-party protocol, where one party (the *constructor*) creates the garbled circuit, and the other party (the *evaluator*) can evaluate the circuit. As the tokens do not reveal their corresponding plain value, and all intermediate values in the evaluation process are also tokens maintaining the hidden semantics, the evaluator can not gain any useful information. However, it is important that the evaluator only learns the tokens corresponding to his inputs. If he also knew the tokens for his negated inputs, he could infer the constructor's inputs. As we also require that the constructor does not learn the evaluator's inputs, the tokens corresponding to his inputs have to be transferred in an oblivious way (see Section 2.3.5). The schematic of the protocol is shown in Figure 2.3.2.

Creating and Evaluating a Garbled Circuit Evaluating a garbled circuit is very similar to evaluating a circuit with the help of truth tables. Figure 2.3.3 shows an AND node and Table 2.3.1 shows the corresponding

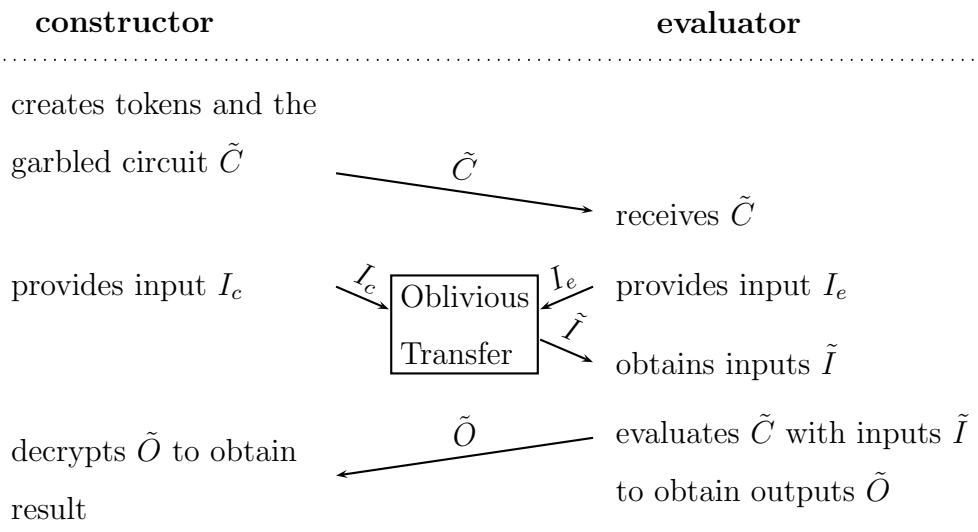


Figure 2.3.2: Schematic of the two party SFE protocol

truth table. A truth table lists the output of the gate for all possible input configurations. Let the inputs of this node be $e_1 = 0$ and $e_2 = 1$. Then the third row of the truth table tells us that the output for this particular input configuration is 0.

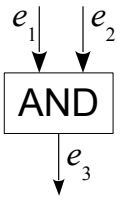


Figure 2.3.3: Node with two inputs e_1 and e_2, which outputs $e_3 = e_1 \wedge e_2$.

input e_1	input e_2	output e_3
0	0	0
0	1	0
1	0	0
1	1	1

Table 2.3.1: Truth table for AND node in Figure 2.3.3

Creating a garbled circuit: The first step in generating a garbled circuit is to replace the plain values in the truth table with random tokens. Let $t^0_{e_i} \in \{0,1\}^k$ be a randomly generated bit string of length k with k being the security parameter. It is used as a token for edge e_i representing the plain

value 0 and $t^1_{e_i}$ is the token representing 1, respectively. Table 2.3.2 shows the garbled truth table for the AND node.

input e_1	input e_2	output e_3
$t^0_{e_1}$	$t^0_{e_2}$	$t^0_{e_3}$
$t^0_{e_1}$	$t^1_{e_2}$	$t^0_{e_3}$
$t^1_{e_1}$	$t^0_{e_2}$	$t^0_{e_3}$
$t^1_{e_1}$	$t^1_{e_2}$	$t^1_{e_3}$

Table 2.3.2: Garbled truth table for AND node in Figure 2.3.3

Obviously, this kind of obfuscation is not enough to hide the plain value of the output edge, since the value that appears three times in the output column of the garbled truth table must be the one representing 0.

In the next step of generating a garbled circuit the constructor encrypts the tokens for the output in the truth table using the tokens of the corresponding inputs as the encryption keys. Table 2.3.3 shows the encrypted outputs, with $\mathrm{Enc}_{t^a_{e_j}}(t^b_{e_i})$ meaning that the token $t^b_{e_i}$ is encrypted with the secret key $t^a_{e_j}$.

input e_1	input e_2	output e_3	encrypted output
$t^0_{e_1}$	$t^0_{e_2}$	$t^0_{e_3}$	$\mathrm{Enc}_{t^0_{e_1}}(\mathrm{Enc}_{t^0_{e_2}}(t^0_{e_3}))$
$t^0_{e_1}$	$t^1_{e_2}$	$t^0_{e_3}$	$\mathrm{Enc}_{t^0_{e_1}}(\mathrm{Enc}_{t^1_{e_2}}(t^0_{e_3}))$
$t^1_{e_1}$	$t^0_{e_2}$	$t^0_{e_3}$	$\mathrm{Enc}_{t^1_{e_1}}(\mathrm{Enc}_{t^0_{e_2}}(t^0_{e_3}))$
$t^1_{e_1}$	$t^1_{e_2}$	$t^1_{e_3}$	$\mathrm{Enc}_{t^1_{e_1}}(\mathrm{Enc}_{t^1_{e_2}}(t^1_{e_3}))$

Table 2.3.3: Garbled truth table with encrypted outputs for AND node in Figure 2.3.3

The important observation is that the evaluator, given the encrypted outputs and the tokens of one input configuration, can only decrypt the corresponding output token (and nothing else). Thus, he will only learn one

input configuration for the subsequent nodes in the circuit, and eventually one circuit output configuration.

That said, there is still one issue that may allow the evaluator to infer the plain value of the decrypted output token. Specifically, knowing the truth table for the node and the position of the entry of the encrypted output tokens, the evaluator can infer the corresponding plain value, *e.g.*, if the evaluator can decrypt the third entry of the encrypted table in Table 2.3.3, then the plain value of the output must be 0 as the third entry of the truth table for an AND node is 0 (see Table 2.3.1).

In the final step of generating a garbled circuit, the constructor randomly permutes the encrypted output tokens. Table 2.3.4 shows an example permutation for our AND node.

input e_1	input e_2	output e_3	encrypted output	permuted output
$t^0_{e_1}$	$t^0_{e_2}$	$t^0_{e_3}$	$\mathrm{Enc}_{t^0_{e_1}}(\mathrm{Enc}_{t^0_{e_2}}(t^0_{e_3}))$	$\mathrm{Enc}_{t^0_{e_1}}(\mathrm{Enc}_{t^1_{e_2}}(t^0_{e_3}))$
$t^0_{e_1}$	$t^1_{e_2}$	$t^0_{e_3}$	$\mathrm{Enc}_{t^0_{e_1}}(\mathrm{Enc}_{t^1_{e_2}}(t^0_{e_3}))$	$\mathrm{Enc}_{t^1_{e_1}}(\mathrm{Enc}_{t^1_{e_2}}(t^1_{e_3}))$
$t^1_{e_1}$	$t^0_{e_2}$	$t^0_{e_3}$	$\mathrm{Enc}_{t^1_{e_1}}(\mathrm{Enc}_{t^0_{e_2}}(t^0_{e_3}))$	$\mathrm{Enc}_{t^0_{e_1}}(\mathrm{Enc}_{t^0_{e_2}}(t^0_{e_3}))$
$t^1_{e_1}$	$t^1_{e_2}$	$t^1_{e_3}$	$\mathrm{Enc}_{t^1_{e_1}}(\mathrm{Enc}_{t^1_{e_2}}(t^1_{e_3}))$	$\mathrm{Enc}_{t^0_{e_1}}(\mathrm{Enc}_{t^1_{e_2}}(t^0_{e_3}))$

Table 2.3.4: Garbled truth table with encrypted and permuted outputs for AND node in Figure 2.3.3

The *garbled circuit* consists of the permuted encrypted output tables of all nodes in the circuit. Algorithm 1 shows the pseudocode for generating a garbled circuit.

Evaluating a garbled circuit: The evaluator needs to know the garbled circuit $\tilde{C}$, the Boolean circuit C and the tokens corresponding to both parties inputs, which are the garbled inputs $\tilde{I}$, to be able to evaluate the circuit. As the nodes in the circuit are ordered topologically, such that the i-th node has no inputs that are an output of a successive node, the evaluator can iterate over all nodes and decrypt one entry of the garbled table, which

Algorithm 1 Algorithm performed by the constructor to create a garbled circuit

Input: Boolean circuit C

Output: Garbled circuit $\tilde{C}$

for all edge e_i in C **do**

create random token $t_{e_i}^0$ and $t_{e_i}^1$

end for

for all node n_i in C **do**

$tt_i \leftarrow$ encrypt and permute the column for the output tokens of the truth table of node n_i

$\tilde{C}[i] \leftarrow tt_i$

end for

subsequently becomes the input token of successive nodes. Algorithm 2 shows the pseudocode for evaluating a garbled circuit.

The question might arise as to how the evaluator may identify the entry in the garbled table which decrypted correctly. One possible solution is to use a *special private-key encryption* as described in [98, Section 3.1]. The idea is to concatenate the token with additional information before encryption. After decryption, this additional information is only present if the correct keys were used, otherwise, it will decrypt to some random value.

However, there is a more efficient way. The previously mentioned approach requires on average 2 1/2 decryptions to find the right entry. The *Point-and-Permute* technique (see Section 2.3.4) allows the evaluator to identify the correct entry straight away by adding random permutation bits to the encrypted values in the garbled table.

2.3.4 Efficiency Considerations

This technique introduces a significant amount of complexity. Every plain text bit is expanded by a factor of the order of the security parameter; addi-

Algorithm 2 Algorithm performed by the evaluator to evaluate a garbled circuit

Input: Garbled circuit $\tilde{C}$, Boolean circuit C, garbled inputs $\tilde{I}$

Output: Garbled outputs $\tilde{O}$

```
for all node n_i in C do
    t_i ← decrypt C̃[i] with node n_i's input tokens
    if output of n_i is output of C then
        Õ.add(t_i)
    end if
    save t_i as input for subsequent nodes
end for
```

tionally, every primitive bit operation is replaced by several comparatively expensive encryptions. Over the last few years several techniques have been proposed to reduce the complexity of garbled circuits.

Encryption Function The dominating factor of the computation costs for the *constructor* is the encryptions for generating the garbled tables and the decryptions for the *evaluator* respectively. Therefore, choosing an appropriate encryption function is pivotal for good performance.

Pinkas *et al.* [119] showed that, using the encryption scheme:

$$\mathrm{Enc}_{t_{e_1},t_{e_2},s}(t_{e_3}) = t_{e_3} \oplus \mathrm{KDF}(t_{e_1}, t_{e_2}, s),$$

with s being a nonce, (an arbitrary number used only once in the protocol), and KDF being a key derivation function[1]. One can realise the KDF using correlation robust hash functions. In their implementation Pinkas *et al.* used one SHA-256 invocation per KDF.

[1] A key derivation function derives one or more secret keys from a secret value such as a master key. In this case, if two nodes in the circuit have the same inputs, then the tokens used as keys for encryption will be the same, as well. The KDF will then derive from those tokens and a nonce a unique encryption key.

Modern CPU now support the AES-NI [128] instruction set, which offers hardware acceleration for computing AES encryption. Kreuter *et al.* [91] achieved a circuit computation time reduction of at least 20 % by using AES rather than the SHA-256 hash function.

Point-and-Permute It takes the evaluator on average $2\frac{1}{2}$ attempts to find the entry in the garbled truth table that decrypts correctly with his input tokens. Malkhi *et al.* [101] proposed a technique called *Point-and-Permute* to enable the evaluator to directly identify the right entry, with no extra costs.

The idea is that, during construction of the garbled circuit, the constructor assigns a random permutation bit to each token such that the two tokens for each edge have opposing permutation bits. He then uses the permutation bits of the tokens of the input edges to permute the garbled table. As the permutation bits are chosen at random, the permutation of the garbled table is thus random as well.

Garbled Row Reduction This technique reduces the size of the garbled table that has to be transferred, thereby reducing the communication complexity. It was first presented in [109]. The observation is that, instead of choosing both tokens for the output edge at random, we can choose one token to be a function of one configuration of the tokens of the inputs. Therefore one entry of the garbled table does not have to be transmitted, leading to a reduction in communication costs of 25%.

Free XOR In [90] Kolesnikov *et al.* showed a construction that enables the evaluation of XOR nodes *for free*. Instead of choosing all tokens at random, the two tokens of an edge have a specific relationship. The construction is as follows: choose $t^0_{e_1}, t^0_{e_2}$ and R at random. Then set $t^0_{e_3} = t^0_{e_1} \oplus t^0_{e_2}$, and $\forall i : t^1_{e_i} = t^0_{e_i} \oplus R$, with $\oplus$ denoting the XOR operation.

Now

$$t^0_{e_3} = t^0_{e_1} \oplus t^0_{e_2} = (t^0_{e_1} \oplus R) \oplus (t^0_{e_2} \oplus R) = t^1_{e_1} \oplus t^1_{e_2},$$

and

$$t^1_{e_3} = t^0_{e_1} \oplus (t^0_{e_2} \oplus R) = t^0_{e_1} \oplus t^1_{e_2} = (t^0_{e_1} \oplus R) \oplus t^0_{e_2} = t^1_{e_1} \oplus t^0_{e_2}.$$

Therefore the evaluator can compute the token of edge e_3 without the need of a garbled table, and with negligible computation costs.

The same technique can also be applied to XNOR nodes.

Reducing the number of non-XOR and non-XNOR nodes in a circuit leads to a decrease in computation and communication costs. In [145] Yu *et al.* and in [119] Pinkas *et al.* showed optimisation techniques to eliminate NOT and constant input gates from Boolean circuits. Techniques to minimise circuits with free XOR and efficient circuit constructions of frequently used circuit building blocks are shown in [133].

Pre-Computation vs. Pipelining The classical approach to run a SFE protocol is that, firstly, the constructor creates the garbled circuit, secondly, he then sends it to the evaluator, and finally, the evaluator starts to evaluate the circuit.

Huang *et al.* [78] proposed another approach, where construction and evaluation of the circuit is interwoven. Their main observation was that, once the constructor has finished computing the garbled table of a node, the evaluator may immediately start evaluating that node, as the circuit is in a topological order. That is, no node depends on the outputs of subsequent nodes. The advantage of this technique is that the constructor does not need to store the whole garbled circuit in memory; in addition, the overall running time (measured from the beginning of garbeling to the end of evaluating) is shortened.

However, if the intended use-case requires a short online phase, *i.e.*, the time from providing the inputs until obtaining the outputs, the classical approach performs better, as the garbled circuit can be precomputed.

2.3.5 Oblivious Transfer

So far we have not tackled the problem of how the evaluator learns the tokens corresponding to his inputs - specifically under the dual constraints that the constructor does not learn his inputs, and that the evaluator does not learn the tokens corresponding to his negated inputs.

The tool of choice for this problem is called an *Oblivious Transfer* (OT) protocol. The idea of OT goes back to Rabin [122]. He proposed a protocol where a sender sends a message to the receiver with probability 1/2, while the sender remains oblivious as to whether the receiver received the message or not.

A slightly different OT protocol is needed for our problem. Even *et al.* [52] transformed Rabin's idea into a *1-out-of-2 OT* protocol. Here the sender has two secrets, whilst the receiver only learns one secret. The protocol ensures that the sender stays oblivious to which secret was chosen by the receiver; Notably, these are exactly the properties we need to transfer the tokens (corresponding with the evaluators inputs) from the constructor to the evaluator.

More efficient 1-out-of-2 OT protocols were proposed by Naor and Pinkas [108] and Aiello *et al.* [5]. However, as these protocols rely on asymmetric cryptography, they are still comparatively expensive and can significantly slow down garbled circuit based protocols with large inputs.

Extending OT Efficiently

Ishai *et al.* [79] showed how to reduce any n 1-out-of-2 OTs of l-bit strings to k 1-out-of-2 OTs of k-bit strings plus a small additional overhead, with security parameter k.

Additionally, as the k 1-out-of-2 OTs are independent of the sender's secrets and the receiver's choices, those OTs can be run in a precomputation phase further reducing the running time of the online phase.

2.3.6 Two-party Secure Function Evaluation (SFE) Protocols

In this thesis we focus on SFE protocols for two parties. That is, we have one constructor creating the garbled circuit, and one evaluator evaluating it.

However, SFE using garbled circuits is not limited to the two-party case. There are protocols that allow for more than two parties, e.g. [109].

The protocol for the two-party case, schematically depicted in Figure 2.3.2, runs as follows:

- The constructor generates a garbled circuit representation $\tilde{C}$ of the function f as described in Section 2.3.3 and sends $\tilde{C}$ to the evaluator.
- The constructor sends the tokens corresponding to his inputs to the evaluator in the clear. The tokens corresponding to the evaluators' inputs are transmitted using an oblivious transfer protocol.
- Now, the evaluator can evaluate the garbled circuit $\tilde{C}$ using the garbled inputs. He obtains the garbled outputs.
- Finally, the garbled outputs are converted into the plain outputs for the respective party. The constructor learns his outputs by receiving the respective garbled outputs from the evaluator (in fact, it is sufficient to just send the permutation bits). As he created these garbled values, he is able to interpret them correctly. For the evaluator to be able to learn his outputs, the constructor has to send helper information; for instance, he can send a mapping from permutation bits of the evaluator's garbled outputs to the corresponding plain values.

Semi-Honest Adversaries

Yao's protocol, as described in Section 2.3.6, is secure against *semi-honest* adversaries [98]. A semi-honest adversary follows the protocol specification

correctly but attempts to learn additional information by analysing the transcript of messages received during the protocol execution.

The security of the protocol is *computationally* bounded. That is, given unlimited computation power, the evaluator could break the garbled circuit. However, by choosing the security parameters appropriately, one can assure that a realistic attack is infeasible.

Covert or Malicious Adversaries

The trust assumption for semi-honest parties might not be true for an intended use-case. Yao's protocol can be extended to protect against more powerful adversaries. There are extensions to protect against both *covert* adveraries [6, 66] and against *malicious* adversaries [91, 97]. Whereas both types of adversaries are allowed to deviate from the protocol as they wish, the covert adversary also does not want to be caught doing so (or the honest party must have a good chance of detecting the covert adversary cheating).

Introducing additional security measures to account for more powerful adversaries comes with extra costs. As the appropriate choice of protection against adversaries depends on the specific use case, we will assume the semi-honest case throughout the thesis for garbled circuit based protocols. However, we want to point out that our solutions are not limited to the semi-honest case. By using one of the aforementioned protocols, one can protect against more powerful adversaries.

In the next chapter we present a new implementation of the two-party SFE protocol. By focussing on memory consumption, we are able to significantly improve the performance compared with previous implementations.

Chapter 3

Memory Efficient Secure Two-Party Computation

In this chapter we present a new implementation of Yao's garbled circuit two-party SFE protocol, augmented by several optimisations; these result in lower memory consumption and significantly better performance compared with previous frameworks.

More specifically, we improve the memory footprint of Yao's SFE protocol by reducing the memory consumption of the oblivious transfer extension protocol (Section 3.3.1), we improve the *maximum working set* technique (Section 3.3.2) and we divide the circuit into sub-circuits (Section 3.3.3). As described in Section 3.1 and summarised in Table 3.1.1, most previous frameworks have a memory consumption which is linear in the size of the evaluated circuit.

We also provide an implementation with performance improvements for base oblivious transfers using multi-threading (Section 3.4.1), and for garbling the circuit using a cache for both circuit descriptions and the communication (Section 3.4.2).

As shown in [83] and implemented in the framework of [91], it is sufficient to just hold the intermediate values in memory that are needed later on, called the *working set*. For example, the maximum size of the working

set of a Karatsuba multiplication of two 128 bit values is 1,074 whereas the circuit has 57,000 gates. In VMCrypt [100], the required memory is dependent upon the way a programmer both creates and decorates circuit components; furthermore the framework of [91] requires additional overhead in the online phase to manage a usage counter and free unused memory (see Section 3.3.2 for details). The memory consumption of our engine has no additional overhead in the online phase.

In this regard, by combining the *maximum working set* technique with the division of a circuit into several repeatedly executed sub-circuits of [78], we achieve a significantly smaller memory footprint for the protocol execution than either technique on its own.

In Section 3.6 we demonstrate that our implementation is substantially more efficient than previous frameworks. As applications, we consider secure evaluation of the Hamming distance, fast multiplication, and computing the minimum. Moreover, we give performance results on both securely computing the AES block cipher, and (for the first time) results on secure evaluation of the ultra-lightweight block cipher PRESENT.

As many previous frameworks do, we provide the source code of our implementation as open source software, to both foster future works and enable a fair performance comparison. The code is available for download at `https://github.com/encryptogroup/me-sfe`.

This chapter represents joint work with Thomas Schneider from the European Center for Security and Privacy by Design (EC SPRIDE), Technische Universität Darmstadt.

3.1 Introduction

Secure two-party computation, often called secure function evaluation (SFE), allows two mutually mistrusting parties to compute an arbitrary function on their private inputs, without revealing any information about their inputs

beyond the function's output. Although the real-world deployment of SFE was believed to be very expensive for a relatively long time, the cost of SFE has been dramatically reduced in the recent years – thanks to many algorithmic improvements and automatic tools, as well as faster computing platforms and communication networks.

SFE enables a large variety of privacy-preserving applications; to name but a few, these include electronic auctions [109], data mining [95], or biometric identification [17, 77].

Although other approaches exist, most practical applications of SFE, (including the ones listed above), are based on Yao's garbled circuits technique [144], for which many improvements have been proposed (we give a summary in 3.2.1). In this chapter we focus on secure two-party computation based on garbled circuits in the semi-honest adversary model. In this model, the adversary is assumed to be "honest-but-curious", *i.e.*, he honestly follows the protocol specification, but tries to learn additional information from the messages seen. Although this adversary model is very weak, it allows the construction of highly efficient protocols for many application scenarios, *e.g.*, for constructing privacy-preserving protocols that protect against attacks by insiders or future break-ins.

We strongly believe that pushing the performance limits of such protocols is essential in order to make secure computation a practical alternative.

Many application scenarios require a low memory footprint, *e.g.*, privacy-preserving applications on smartphones [75, 76]; generating garbled circuits in resource-restricted trusted hardware [82]; evaluating garbled circuits with a hardware accelerator [83]; or securely evaluating large functionalities in cloud computing scenarios [28]. Our framework has a smaller memory footprint than any previously published framework and, combined with its excellent performance, represents the best candidate to enable such applications.

Frameworks for secure two-party computation in the semi-honest adversaries setting can be classified into either the traditional compilation

paradigm, (where the function to be computed is first compiled), or the 'on-the-fly' paradigm (that generates circuits gate by gate from a library). The compilation paradigm is used in Fairplay [11, 101] and TASTY [71]. The 'on-the-fly' paradigm is used in the FastGC framework [77] and VMCrypt [100]. We provide the best of both worlds by compiling and optimising sub-circuits once and dynamically composing these sub-circuits on-the-fly.

3.1.1 Related Work

Several frameworks for SFE with different properties have been proposed to achieve performance improvements (as summarised in Table 3.1.1). These frameworks allow an application developer to describe the functionality required for secure computation at a high level, abstracted from the details of the underlying protocol. Fairplay [11, 101] allows the functionality to be computed to be described in a high-level language which is compiled into a Boolean circuit in an offline pre-computation phase. This compilation allows global optimisations, such as eliminating dead code. Subsequently, TASTY [71] partitioned the garbled circuit protocol such that most of the expensive operations (w.r.t. both, communication and computation) are performed in the pre-computation phase. VMCrypt [100] introduced the concept of streaming, *i.e.*, the garbled circuit is generated gate by gate, and directly streamed into the network, to enable smaller memory footprints. Previously the whole garbled circuit was created and held in memory before it was sent to the other party. VMCrypt achieves a smaller memory footprint by allowing the composition of the circuit by dynamically constructing and deconstructing sub-circuits. However, VMCrypt instantiates a new object for each gate, and so its performance suffers from the additional overhead of garbage collection. Similarly, in FastGC [78] the circuit is not compiled, but composed from sub-circuits, and dynamically generated within a library. Furthermore, a new object is created for each gate of the sub-circuit, which could be freed by the garbage collector when no longer used. GCParser [103]

Framework	Compilation or Streaming	Memory Footprint	GC/ UC
Fairplay [11, 101]	C	$\mathcal{O}$(circuit size)	none
TASTY [71]	C	$\mathcal{O}$(circuit size)	none
VMCrypt [100]	S	$\mathcal{O}$(max(size of sub-circuit))	GC
FastGC [78]	S	$\mathcal{O}$(max(size of sub-circuit))	GC
GCParser [103]	both	$\mathcal{O}$(max(size of sub-circuit))	GC
[91] (cluster)	both	$\mathcal{O}$(mws(circuit))	UC
This Work	both	$\mathcal{O}$(max(mws(sub-circuit)))	none

Table 3.1.1: Frameworks for secure two-party computation in the semi-honest adversaries setting. GC: garbage collection, UC: usage counter, mws(x): maximum working set of x.

extended the FastGC framework to read in a file which describes the way pre-defined sub-circuits should be put together; it also requires memory linear in the size of the sub-circuits. Most recently, the framework of [91] implemented GC-based secure function evaluation in the malicious setting, by exploiting the high degree of parallelism available in a cluster. In this framework, each gate carries a usage counter, such that memory can be freed after the last use of the gate; however, this requires additional overhead in the online phase. Nonetheless, the *maximum working set*, (the outgoing wire labels of the gates, that are either an output to the circuit or an input to subsequent gates), is smaller – and, in most cases, significantly smaller than the circuit. We extend their ideas for memory-efficient secure evaluation of garbled circuits by modifying the *maximum working set* idea to eliminate the usage counter and applying it to sub-circuit optimisation.

Furthermore, a compilation technique for memory-efficient on-the-fly generation of circuits from Fairplay's high-level description language was proposed in [104]. Alternatively, circuits can also be compiled from ANSI

C programs, as shown in [74]. The FastGC framework [78] was recently extended to process circuit descriptions, which are composed from hard-coded circuit building blocks [103].
However, these techniques do not minimise the amount of memory needed during secure evaluation of the circuit.

3.2 Preliminaries

3.2.1 Yao's Garbled Circuit Protocol

Yao's garbled circuit protocol [144] allows two parties, namely a server (constructor) and a client (evaluator), to jointly compute a function f represented as Boolean circuit on their respective private inputs x and y. At a very high level, the constructor creates an encrypted (garbled) version of f which is then sent to the evaluator, who evaluates the function under encryption. The server also has to send encrypted versions of the private inputs to the evaluator. For the evaluator's inputs this is done with a subprotocol, called oblivious transfer, such that the constructor does not learn the evaluator's inputs (see below for details).

We provide a more detailed description of Yao's protocol in Section 2.3.3.

The following optimisations of garbled circuits and oblivious transfer are used in today's most efficient implementations of Yao's protocol, including [17, 71, 78, 91, 100] and our implementation.

Garbled Circuit Optimisations

The point-and-permute technique [109] represents each wire label as a symmetric t-bit key and a permutation bit π, where t is a symmetric security parameter. The permutation bits of a gate's input wires are used as an index to denote which table entry needs to be decrypted. The free XOR technique [90] allows computation of garbled XOR and XNOR gates without communica-

tion (no garbled table is needed), and only negligible computation. Thus, the dominating factor for the complexity of a circuit is the number of non-XOR gates. Further, the garbled row-reduction technique [109, 119] allows reduction of the size of the garbled table by one entry.

We described these optimisation techniques in more detail in Section 2.3.4.

Oblivious Transfer

In m-parallel Oblivious Transfer (OT) of ℓ-bit strings, denoted as OT_ℓ^m, the chooser inputs a vector of choice bits r_i, with $i = 1, \ldots, m$ and the sender inputs a vector of pairs of ℓ-bit strings $(x_0, x_1)_i$, $i = 1, \ldots, m$. At the end of the protocol, the chooser learns the selected strings $x_{r_i,i}$, but nothing about the other strings $x_{1-r_i,i}$ whereas the sender learns nothing about the choices r_i.

When using OT to transfer the input labels in Yao's Garbled Circuit protocol, $\ell = t+1$ (t is the protocol-wide symmetric security parameter, and $+1$ because of the extra permutation bit needed for the point-and-permute optimisation technique), whereas m corresponds to the number of input bits provided by the evaluator which can be large. Using OT extensions of [79] it is possible to reduce a large number of OTs to a small number of only k OTs, where k is a security parameter. These remaining k base OTs are implemented with an efficient OT protocol which requires $\mathcal{O}(k)$ public-key operations. Naor *et al.* [108] provide such an OT protocol.

In Section 3.4.1 we give implementation improvements for the base OTs of [108] and in Section 3.3.1 we show how the OT extension of [79] can be implemented with a small memory footprint.

3.3 Secure Evaluation of Garbled Circuits with Less Memory

3.3.1 Extending OTs with Low Memory Footprint

A large number of m parallel OTs of ℓ-bit strings, OT_ℓ^m can be reduced to a small number of only $k \ll m$ OTs of k-bit strings, OT_k^k, using OT extensions of [79] as implemented in [78]. The original protocol of [79] needs only two messages beyond the messages of the base OTs, but memory linear in m.

We reduce the memory requirement of this protocol by re-ordering its messages in the following ways.

To begin, the OT extension construction of [79] proceeds in two steps. Firstly, the large number of m parallel OTs of short ℓ-bit strings are reduced to k parallel OTs of long m-bit strings, cf. [79, Fig. 1]. These OTs are implemented using k parallel OTs of short k-bit keys that are then stretched into longer m-bit masks using a pseudo-random generator (PRG), cf. [79, Fig. 3].

A pseudo-random generator (PRG) maps a random seed to a longer pseudo-random string, *i.e.* $\mathrm{PRG}(k) : \{0,1\}^k \mapsto \{0,1\}^*$. A very efficient and standard way to implement a PRG is to successively concatenate the output of a pseudo-random permutation (PRP) keyed with a counter, *i.e.*, $\mathrm{PRG}(k) = \mathrm{PRP}_k(0)||\mathrm{PRP}_k(1)||\ldots$. In practice, the PRP can be instantiated with a block cipher which operates on blocks of M bits, *e.g.*, in our implementation we use AES-128 with $M = 128$. Now, in order to reduce the memory footprint, the OT extension construction of [79] can be easily split into smaller blocks, where each block performs M parallel OTs. The overall protocol is shown in Figure 3.3.1. To simplify presentation, we assume w.l.o.g. that m is a multiple of the block size, $m = BM$ (otherwise, the last messages are shorter), and that $\log B \leq M$. $\mathbf{T}$ denotes an $M \times k$ bit matrix, $\mathbf{T}^i$ its i-th column and $\mathbf{T}_j$ its j-th row. $G : \{0,1\}^M \times \{0,1\}^k \to \{0,1\}^M$ is

a PRP; $H : \{0,1\}^{\lceil \log m \rceil} \times \{0,1\}^k \to \{0,1\}^\ell$ is a random oracle which can be implemented using a cryptographic hash function (in our implementation we use SHA-1).

Overall, our modified protocol can be seen as operating on a stream of data which is processed in chunks of size M.

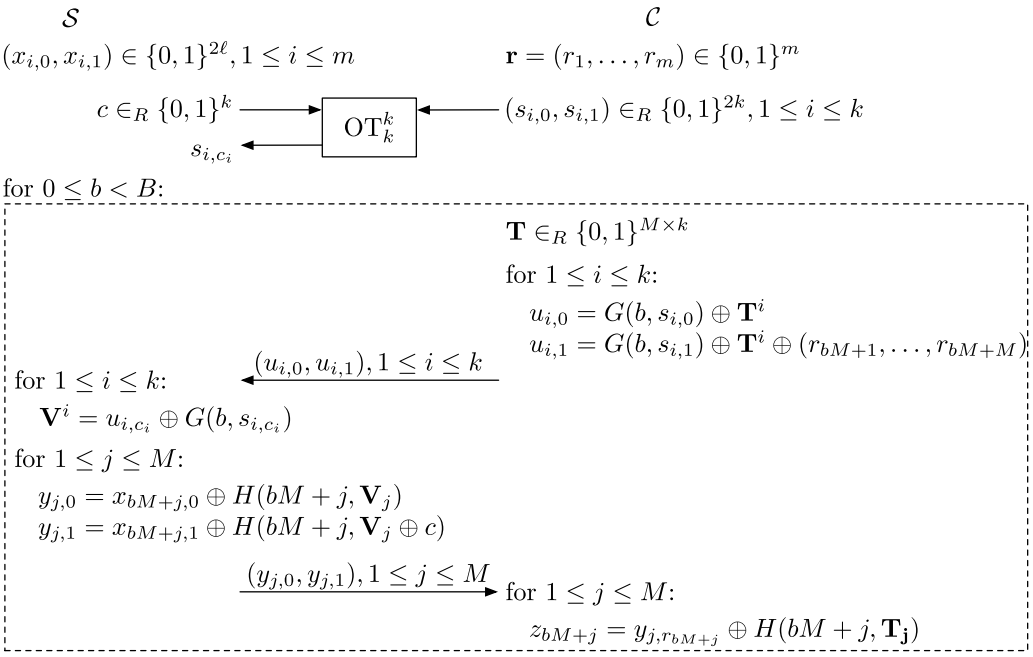


Figure 3.3.1: Extension of OT_ℓ^m with low memory footprint. W.l.o.g. $m = BM$ for B blocks of size M. G is a pseudo-random permutation and H is a random oracle.

Security and Correctness The only modification we applied to the original construction of [79] was the reordering of the messages sent into B rounds of communication. For semi-honest parties, this does not reveal any additional information. Therefore, the correctness and security of our optimised protocol in the semi-honest setting directly follows from the correctness and security of the original construction, as proven in [79].

Performance The computation and communication complexity of our protocol is identical to that of the original construction of [79]. In contrast to the implementation in [78], (which has a constant number of communication rounds, but requires memory linear in the total number of OTs m), the memory consumption of our protocol is constant (for fixed block size M), but requires m/M rounds of communication. We give performance benchmarks for the improved OT extensions in Section 3.6.1.

3.3.2 Streaming Circuits and Garbled Circuits with a Small Memory Footprint

As described in Sections 3.1 and 3.1.1, previous frameworks for secure two-party computation (in the semi-honest adversaries setting) store the outgoing wire labels of each gate in memory, and hence require memory linear in the size of the evaluated circuit. More recent frameworks [78, 100, 103] require memory only linear in the size of the sub-circuits, but these frameworks suffer from the low performance of memory management of many small objects (one object for each gate) and garbage collection.

During creation and evaluation of a (sub) circuit, only those wires that are used in the future need to be held in memory – that is, only those that are either an output wire of the circuit, or those used as input wires into a later gate. This set of wires is called the *working set*. Thus, the minimum size of the memory required to compute the circuit is defined by the maximum working set. As described and implemented in [91], one strategy to keep track of the working set is to annotate each wire with a usage counter that is decremented on each use. When the counter reaches zero, the wire is deleted from the memory. Similarly, sub-circuits are dynamically constructed and deconstructed in VMCrypt [100], which creates additional overhead in the online phase.

We use a different approach – specifically where we shift the management of the working set from the online phase to the compilation phase in order to keep the online phase as lean as possible. In particular, the online phase of our approach neither requires a usage counter, nor the allocation of memory for each gate. The compilation of any given Boolean circuit has to be done once only, and can be reused for unlimited SFE protocol executions. We therefore not only reduce the memory requirements, but also keep the computations as lean as possible by omitting both handling of usage counters and dynamic object construction/deconstruction. During the compilation phase, the compiler determines the maximum size S of the working set, and stores this alongside the circuit description. The compiler allocates a slot ID (starting from 0) to each input wire of the circuit. Next, the circuit description is generated as an ordered list of gates, where each gate is described as a tuple (output slot ID; input slot IDs; truth table). Whenever a slot ID is used for the last time, it is added to a list of available slot IDs, and can be re-used as the output slot ID of a later gate. Note that all this is done in the offline (compilation) phase. In the online phase, an array with S slots is allocated where each slot can hold a wire label. Then, the gates are read one-by-one from the circuit description: the gate's input labels are taken from the slots given by its input slot IDs and the output label is stored to the slot specified by the gate's output slot ID. We internally keep a counter of the gate ID which is used to construct the garbled tables.

We emphasise that the compilation of a function need only be completed just once, since the resulting circuit is independent from the inputs, and can therefore be reused.

In future work, the compiler can be extended to re-arrange the order of the gates in order to reduce the size of the working set as described in [83]. However, as noted in [91], determining a topological order of the circuit with the minimum size of the working set is known to be an NP-complete problem.

We further note that, in principle, generation of garbled circuits can be implemented such that the amount of memory is constant, by pseudo-randomly deriving the wire labels from the gate ID (as described in [82]). However, this is essentially a time-memory tradeoff, and cannot easily be combined with the highly efficient free-XOR technique.

3.3.3 Sub-Circuit Compilation

A design goal of our framework was to keep the online phase of the circuit evaluation as lean as possible. Due to the topological ordering of the circuits, the engine never has to hold more than one gate description in memory. Once a gate is processed, the information can be discarded (except for the intermediate wire labels). The circuit evaluation engine reads the circuit description from a file with a format similar to Fairplay. In order to re-use sub-circuits, we use slot IDs to index the wires in the sub-circuit. A slot ID can be seen as a virtual register that can hold a wire label and can also be re-used. For the gate ID, (which needs to be unique in the overall circuit, as it is used for encrypting the non-linear gates), we use a counter which is incremented for each non-linear gate. A gate is described by its output slot ID, its input slot IDs, and its truth-table. An example is shown in Figure 3.3.2, which describes the one-bit comparison circuit from [88]. We also support a binary file format, which is more efficient to read using our engine.

The circuit to be computed often consists of several calls to the same sub-routine. For instance, AES encryption consist of three sub-routines: S-Box, AddRoundKey, and MixColumns, which are repeatedly called. The number of invocations are shown in Table 3.3.2.

Our framework allows for sub-circuits to be reused in a similar fashion to [78], except that we do not instantiate a new gate object in each invocation of the sub-circuit. Overall, for AES we do not create the entire circuit with 24,720 gates, but only 3 sub-circuits for the sub-functions listed above, with

sub-routine	invocations
S-Box	160
AddRoundKey	10
MixColumns	9

Table 3.3.2: Number of invocations of the sub-routines in AES encryption

```
inputsCreator: 0    // creator's input is r0
inputsEvaluator: 1  // evaluator's input is r1
outputsCreator:
outputsEvaluator: 0 // evaluator's output is r0
numberOfRegisters: 2
numberOfGates: 2
0;1,0;1             // r0 = r1 and r0
0;1,0;6             // r0 = r1 xor r0
```

Figure 3.3.2: A one-bit comparison circuit. Comments (from // on) are not part of the input.

a total of just 803 gates. In our implementation the creation of the gate ID is decoupled from the circuit definition – thus ensuring that in every reuse of a sub-circuit, all gates have a unique gate ID, and therefore the security of the underlying garbled circuit protocol, (as proven in [98, 119]), still holds.

We provide a compiler that converts circuits described in the format of [78] into our format.

3.4 High Performance Implementation

We optimise several aspects of the FastGC framework [78], as described in the following. We emphasise that our optimisations do not modify the underlying cryptographic protocols, but only the way they are implemented.

Therefore, and because we work in the semi-honest setting, the proofs of security of the original protocols [98] still hold for our optimisations.

3.4.1 Improved Base OTs

In order to improve the performance of the k base OTs with the OT protocol of [108], we split up the computationally intensive public key operations into multiple threads such that each of the N threads performs k/N base OTs (independent of each other). Furthermore, we also experimented with an implementation of the base OTs over an elliptic curve instead of over $\mathbb{F}_P$ (the latter was used in the implementation of [78]). This resulted in a reduction of the communication complexity, but unfortunately worse runtimes. Although the complexity of the modular exponentiations is significantly reduced due to the smaller key sizes in elliptic curves, the additional computations for arithmetic on elliptic curves outweigh the gains for the faster modular exponentiations in our implementation. Our performance benchmarks for the improved base OT implementations are given in Section 3.6.1.

3.4.2 Caching of Circuits and Communication

In order to improve the performance of garbled circuit evaluation, we cache both circuit descriptions and network packets during garbled circuit streaming – resulting in a corresponding time-memory trade-off as described next.

In some circuits, the same sub-circuits are reused many times (cf. Section 3.3.3). Instead of reading the description of the sub-circuit from a file on every instantiation (or re-generating it as implemented in previous frameworks), we optionally cache its description once in memory. The memory consumption is 32 bytes per cached gate.

Sending both the creator's input wire labels, and the garbled tables immediately after creation (as implemented in [78]) leads to an inefficient use of the network, because of small packet sizes and an unnecessary large num-

ber of packets. By using fixed sized buffers on the communication channels, we greatly improve the performance of the network usage.

In our benchmarks in Section 3.6, we use circuit caching and network buffers of size 9,000 bytes.

3.4.3 Compiler

We also provide a compiler to transfer circuit descriptions in the format of [78] into our format. The compiler determines the maximum working set and assigns the slot IDs accordingly.

3.5 Applications

We compare the performance of our implementation to the performance of previous frameworks using the following applications. We show that choosing a more appropriate circuit representation of the problem results in significantly smaller circuit sizes.

Hamming Weight

Some applications, *e.g.*, privacy-preserving face recognition [113], need to securely compute the Hamming distance $d_H(\vec{a}, \vec{b})$ between two ℓ-bit strings $\vec{a}$, $\vec{b}$. As shown in [78], this can be done by XOR-ing $\vec{a}$ and $\vec{b}$ bitwise and computing the Hamming weight $h(\cdot)$, *i.e.*, the number of "ones" in this binary representation: $d_H(\vec{a}, \vec{b}) = h(\vec{a} \oplus \vec{b})$.

Hamming Circuit of [78]. The authors of [78] propose to use a tree of addition circuits which requires approximately $\sum_{i=0}^{\lceil \log_2 \ell \rceil} \ell \frac{i}{2^i} = \ell(2 - \frac{\lceil \log_2 \ell \rceil + 2}{2^{\lceil \log_2 \ell \rceil}}) \approx 2\ell - \log_2 \ell$ non-XOR gates. For the example of $\ell = 900$ given in their paper this yields approximately 1,790 non-XOR gates.

Improved Hamming Circuit. We use the optimised Hamming weight circuit of [24]. This circuit contains only $\ell - h(\ell)$ non-XOR gates for two ℓ-bit strings as inputs, where $h(\ell)$ is the Hamming weight of ℓ. For $\ell = 900$

this yields $900 - h((1110000100)_2) = 896$ non-XOR gates. This is about half the size of the previous circuit.

Block Ciphers

Oblivious evaluation of a block cipher, where one party provides the key and the other party provides the message and obtains the ciphertext, has many applications as summarised in [119]. These include oblivious pseudo-random functions (OPRFs), with applications to secure keyword searching [56], or secure set intersection [81], blind MACs, and blind encryption.

As noted in [78], the key schedule of the block cipher does not need to be computed securely within the garbled circuit. Instead, the party that knows the key can run the key schedule on the plain key data to both expand it, and to provide the expanded key as input to the protocol.

AES Secure evaluation of AES is commonly used as a performance benchmark for secure computation frameworks, *e.g.*, [71, 78, 91, 119].

AES Circuit of [78]. Excluding the key schedule, AES-128 consists of 10 rounds where in each round 16 S-boxes are evaluated. As shown in [78, Sect. 7], all other operations, *e.g.*, MixColumns and AddRoundKey, can be performed using only free XOR gates. The S-box presented in [78] has 58 non-XOR gates resulting in $58 \times 10 \times 16 = 9{,}280$ non-XOR gates for AES.

Improved AES Circuit. Instead, we implemented the S-box of [25] which consists of only 32 non-XOR gates resulting in a total of $32 \times 10 \times 16 = 5{,}120$ non-XOR gates for AES. We note that the AES circuit implemented in [91] uses the same S-box and has 9,100 non-XOR gates, but including the key schedule.

PRESENT For the applications mentioned above, it might be sufficient to use a block cipher that does not provide the strong security guarantees of AES, but is more efficient to evaluate. An example for such an ultra-

lightweight block cipher is PRESENT with a block length of 64 bit and an 80 bit key. PRESENT consists of 31 rounds where in each round a 4-bit S-box is applied 16 times in parallel. We implemented PRESENT using the S-box representation of [40, 41] which has 4 non-XOR gates. Overall, PRESENT requires $31 \times 16 \times 4 = 1{,}984$ non-XOR gates.

Fast Multiplication

To compare our implementation with FastGC [78] and TASTY [71] for circuits of medium size, we implemented secure multiplication using the fast multiplication method of Karatsuba and Ofman [85]. For multiplication of two 128 bit numbers, this circuit consists of 17,973 non-XOR gates.

Minimum

To compare our implementation with FastGC [78] and VMCrypt [100] for circuits of large size, we implemented a circuit to compute the minimum of 10^6 20-bit numbers (half of the numbers are input by the server and the other half by the client) using a circuit similar to the one described in [100, Fig. 2]. We use the OT extension with low memory footprint described in Section 3.3.1 such that the total memory consumption stays linear in the size of the subset and not in the order of the total number of inputs. The overall circuit has $2 \times (10^6 - 1) \times 20 \approx 40{,}000{,}000$ non-XOR gates.

There are different approaches to compute this functionality, demonstrating the flexibility of our framework. It enables a trade-off between execution time (shown in Table 3.6.4) and memory consumption (shown in Table 3.6.6):

a) The first solution has the lowest OT overhead – by completing the OTs for all inputs first, and then evaluating a large circuit with a working set of size $2 * 10^7$. Obviously, this approach requires the maximum amount of memory (800 MB).

b) As another extreme measure, we can iteratively compare one input each at a time with the previously found minimum value. This approach needs the minimal amount of memory (18.4 MB), yet introduces additional overhead.

c) Our framework allows one to choose an intermediate approach, where we iteratively compute the minimum of a subset of 500 inputs and the minimum of the previous iteration. This sub-circuit has 19,960 non-XOR gates and a working set of 10,022 labels, and is small enough to be cached in memory – the total memory requirement is 21.5 MB.

3.6 Performance Benchmarks

In the following we show that the implementation of our improvements result in substantially better performance than previous frameworks.

3.6.1 Oblivious Transfers

The following performance benchmarks were performed on two Apple computers with a dual core processor each (Intel Core i5 2.5GHz and Core i7 1.8GHz) running MacOS X 10.7.4 and Java 1.6.0_33, connected via 802.11n WIFI.

We observed that, because of the JAVA just in time compiler, the runtime decreases in the first few runs due to compiler optimisations. Therefore, we executed each protocol 1,000 times and took the average. All benchmarks were executed with the default JAVA VM parameters.

The performance of our improved implementation for the base OTs (cf. Section 3.4.1), in comparison with the original implementation of [78], is shown in Table 3.6.3. Thread management introduced additional overhead, therefore the execution time for the two threaded version is not quite half the execution time for the single threaded equivalent. However, running

four threads on a two-core processor reduced the overhead, and we achieve a runtime of just 153 ms compared with the 286 ms of the single threaded version.

single threaded	time
over $\mathbb{F}_P$ [78]	286 ms
over EC	560 ms
multi threaded (over $\mathbb{F}_P$)	time
2 threads	182 ms
4 threads	153 ms

Table 3.6.3: Comparison of base OT implementations.

For the OT extensions (independent of the performance of the base OTs), our improved implementation of the protocol of Section 3.3.1 can evaluate about 400,000 OTs per second; that is, $2.5\mu s$ per OT, a factor of 6 improvement over the $15\mu s$ reported in [78] (which used faster Intel Core Duos E8400 3GHz and a faster local area network).[1] We emphasise that this optimisation is very beneficial for applications where the circuit has many inputs, *e.g.*, for converting from homomorphic encryption to garbled circuits and subsequently finding the minimum – a very common building block in privacy-preserving protocols for biometric matching [17, 71, 77].

3.6.2 Online Time

We changed our benchmark setting for measuring the *online* time – that is, the time from after the connection is established and the base OTs are completed, until the end of the protocol. The following benchmarks were

[1] For completeness we note that [111] claims an actively-secure OT extension that can be implemented at about 500,000 OTs per second based on unpublished optimisations (cf. Appendix A and E in the full version of their paper).

executed on a single iMac A1311 with an Intel Core i3 3GHz processor using the loopback network interface, because a network introduces additional overhead, depending on the type of network, load on the links, etc. and we seek to show the performance of the protocol unaltered by network overhead.

Our improved implementation evaluates about 500,000 non-linear gates per second ($2\mu s$ per gate) on the same host (setting as described above), and about 350,000 non-linear gates per second ($3\mu s$ per gate) over a WLAN (setting as described in Section 3.6.1). In contrast, [78] reported 96,000 non-linear gates per second ($10\mu s$ per gate) over a LAN.[2]

In the following we show that the online time of our framework when evaluated on the same host as described above (*i.e.*, assuming an ideal network) is up to 10 times faster than that of the FastGC [78] framework. The comparison between our implementation and FastGC [78] for the applications described in Section 3.5, both executed on exactly the same machine, is summarised in Table 3.6.4.

Using the optimised Hamming circuit already improves the result of [78] by about 40%. Executed with our implementation, we achieve online times which are more than 10 times faster than the original Hamming circuit evaluated with FastGC.

The results for the AES cipher are similar. Choosing the improved circuit almost halves the online time – but again, the execution with our implementation is approximately 12 times faster than the previously published fastest implementation. Due to its smaller gate count, PRESENT is almost 4 times faster than the original AES circuit, and twice as fast as the improved AES circuit.

The multiplication circuit took 45 ms online time to evaluate in our optimised framework. The same circuit implemented in FastGC [78], and run on the same machine, took 499 ms, – more than 10 times longer. This

[2]In the malicious setting, [88] report 82,000 non-linear gates per second ($12\mu s$ per gate) on a cluster and [111] 20,000 gates per second ($50\mu s$ per gate) over an intranet.

Circuit	non-XOR gates	FastGC [78]	Our Implementation
Original Hamming	1,793	64 ms	8 ms
Improved Hamming	896	39 ms	6 ms
Original AES	9,280	204 ms	27 ms
Improved AES	5,120	113 ms	16 ms
PRESENT	1,984	53 ms	7 ms
Fast Multiplication	17,973	499 ms	45 ms
Minimum	40,000,000	1,250 s	a) 138 s b) 272 s c) 128 s

Table 3.6.4: Comparison of circuit sizes and performance when run on the same machine.

result supports the fact that the improved online time of our implementation, in particular for medium size circuits, is substantially faster than that of [78], even without circuit-specific optimisations. According to [71, Fig. 7], TASTY takes approximately 4,000 ms setup time, and 700 ms online time to evaluate the same circuit on two desktop PCs with Intel Core 2 Duo CPU (E6850) running at 3GHz connected via Gigabit Ethernet. We emphasise that TASTY does not use streaming, but rather pre-computes the OT extensions and generates and transfers the garbled circuit already in the setup phase; as such, the online time in TASTY consists only of the very efficient online OTs of Beaver's construction [8] and garbled circuit evaluation; in contrast, our online time includes the computationally more expensive OT extensions, plus creating and transferring the garbled circuit (which is minimised due to streaming). Overall, for this application, the online time of our improved framework is faster than TASTY by a factor of approximately 16 times, whereas our setup time is as low as 153 ms for the base OTs (cf. Table 3.6.3), *i.e.*, 26 times faster.

We chose three different approaches to compute the minimum. The one with the smallest overhead for the OT protocol (a) has a runtime of 138 s, but the highest memory consumption. The other extreme, iteratively comparing one input each at a time with the previously found minimum (b), needs the minimal amount of memory (18.4 MB) but the maximal total runtime (272 s), as the OT protocol introduces a significant overhead. The last approach (c) realises a trade-off between fast online time and low memory consumption. We iteratively compute the minimum of a subset of 500 inputs combined with the minimum of the previous iteration. This sub-circuit has 19,960 non-XOR gates and a working set of 10,022 labels, and is small enough to be cached in memory – the total memory requirement is 21.5 MB and the total runtime is 128 s. In contrast, when evaluating the circuit for approach (c) on the same machine with FastGC [78], this took approximately 1,250 s and 189 MB memory – specifically more than 9 times longer and 9 times more memory requirement compared with our implementation. According to [100, Fig. 8], VMCrypt takes 44.5 min on a slower CPU (Thinkpad X301 laptop with 3 GB RAM and a 1.6 GHz Intel Core2 Duo processor running Ubuntu Linux over the loopback interface), *i.e.*, about 10 times longer than our approach c), allowing for the fact that our CPU is about twice as fast. We assume that our improved performance stems mainly from the fact that we do not allocate and free many small objects.

3.6.3 Memory Consumption

The memory consumption of a SFE protocol is dominated by the number of garbled labels concurrently held in memory. We showed that all other elements of the protocol execution require only a constant amount of memory.

Table 3.6.5 shows the memory consumption for the applications described in Section 3.5. It shows the total number of gates in each circuit, which coincides with the memory consumption of classical implementations like Fairplay [101] or TASTY [71].

Some circuits can be divided into sub-circuits, that can be re-used. For instance, the block ciphers AES and PRESENT consist of three sub-routines that are called frequently. The combined size of these sub-circuits is significantly smaller than the original circuits, and therefore leads to large memory savings. However, not every problem is divisible and therefore the sub-circuit technique will not always improve memory consumption (see, for instance, the Hamming circuits or the fast multiplication circuit).

The working set technique as described in Section 3.3.2 will always improve the memory footprint, as not reusing any slotID will result in exactly the same memory footprint as the circuit. We achieve significant memory savings for all applications.

Circuit	tot. number of gates	gates of sub-circuits	working set
Orig. Hamming	7,163	7,163	1,800
Impr. Hamming	5,362	5,362	1,800
Original AES	36,720	878	256
Improved AES	24,720	803	256
PRESENT	8,496	145	128
Fast Multipl.	57,072	57,072	1,074
Minimum a)		137,999,862	20,000,002
Minimum b)	137,999,862	414	104
Minimum c)		69,000	10,022

Table 3.6.5: Comparison of the number of gates required to be held in memory for the classical approach, division into sub-circuits, and reduction to working set.

We measured the memory consumption of our Implementation to give an indication as to how our memory saving techniques benefit real programs. However, measuring the memory consumption of a Java program is fuzzy,

since released objects remain on the heap until the garbage collector deletes them, and the garbage collector itself is managed by the Java virtual machine. Thus, the heap contains not only the currently used objects, but also released objects. Therefore the size of the heap will be greater than or equal to the size of the currently used objects. We measured the maximum heap consumption during a protocol run, since this gives an indication of how much memory is needed for a runtime optimal execution. The protocols might run with smaller heap sizes, but then the virtual machine has to invoke the garbage collection more often, resulting in longer runtimes.

Table 3.6.6 shows our measurements for the different applications.

Circuit	FastGC	Our Impl.
Orig. Hamming	28.5 MB	17 MB
Impr. Hamming	26 MB	17 MB
Original AES	20.3 MB	16.9 MB
Improved AES	20.2 MB	18.7 MB
PRESENT	18.9 MB	18.6 MB
Fast Multipl.	74.4 MB	15 MB
Minimum a)		799.7 MB
Minimum b)	189 MB	18.4 MB
Minimum c)		21.5 MB

Table 3.6.6: Comparison of the memory consumption.

It is important to stress that, although the memory consumption of the protocol implemented in a different programming language might be smaller compared with our implementation, the overall trend stays the same.

The memory consumption of FastGC is linear in the total number of gates, whereas in our implementation it is linear in the size of the working set. Although for circuits that are divisible into many small sub-circuits (such as AES or PRESENT) the memory consumption of both implementations

is almost the same, the memory efficiency advantage of our implementation becomes obvious for larger circuits. Compared with FastGC, we achieve a memory consumption reduction by a factor of 5 for Fast Multiplication, and by factor of 8 for Minimum approach c) – indeed, by combining the repeated use of a sub-circuit and then holding the working set in memory, we can evaluate the Minimum circuit with almost 140 million gates using only 21.5 MB of memory.

3.7 Conclusion

In this chapter, we have presented a new implementation of Yao's garbled circuit two-party SFE protocol with significantly better performance than previous frameworks. We focus on efficient use of memory. By decoupling memory demand from the circuit size, and presenting an OT extension with fixed memory consumption overhead, we presented an implementation with low memory footprint.

These improvements enable our implementation to evaluate bigger circuits with less memory. In particular it enables the usage of SFE protocols on devices with limited memory (mobile phones, wearables, internet of things).

We also showed that choosing an appropriate Boolean circuit representation can significantly improve performance. We improved a previously published implementation of Hamming distance by 39% and AES encryption by 45%.

Our implementation decouples the unique gate IDs, which are important for the security of the encryption function, from the circuit description. This enables the framework to re-use parts of the circuits and dynamic circuit sizes.

In the next chapter, we present a solution to the scaling problem – one that uses this framework to evaluate a dynamic circuit with size determined at runtime.

Chapter 4

Conversion of Real-Numbered Privacy-Preserving Problems into the Integer Domain

Secure Multiparty Computation (SMC) enables untrusting parties to jointly compute a function on their respective inputs without revealing any more information than the outcome itself. As we have seen, there are many available algorithms and tools to support SMC, but almost all techniques for SMC support only integer inputs and operations.

We present a *secure scaling* protocol for two parties to map real number inputs into integers without revealing any information regarding their respective inputs. The main component is a novel algorithm for privacy-preserving random number generation. We also show how to implement the protocol (using Yao's garbled circuit technique), and provide an implementation using the memory efficient secure two-party framework we described in Chapter 3.

4.1 Introduction

For the last 30 years the field of privacy-preserving techniques for distributed computation, also called *Secure Multiparty Computation* (SMC), has been growing. It offers solutions for multiple parties to compute functions without revealing their respective inputs to each other. These techniques have come a long way from early theoretical ideas, now providing practical solutions for problems such as electronic voting, auctions, data mining, network management and optimisation.

Almost all secure multiparty computation techniques have a message space consisting of a finite set of integers, and the operations they provide are only defined over the integers. What if one wants to engage in a privacy preserving protocol with real numbers, or floating point approximations? One can either extend a SMC technique to support fixed-point or floating-point arithmetic, or alternatively create a mapping from the inputs into the integer space, and then use conventional SMC. Catrina *et al.* [31] extended the ideas of secure computation based on secret sharing [43] to support fixed-point numbers. It introduces additional complexity, as inputs either have to be converted into the same fixed-point type, or integer and fractional parts of a number have to be treated separately. Fouque *et al.* [54] based their idea on homomorphic encryption; however, an additional invocation of Gauss' algorithm for finding a basis in a 2-dimensional lattice is necessary to recover the results of a computation. This construction limits the amount of additions and multiplication. Franz *et al.* [55] also use homomorphic encryption, but they chose a logarithmic representation of floating-point numbers for fixed maximum relative error. The numbers must necessarily come from some pre-defined interval, and each gets represented by three different parameter. In each subsequent computation, the three parameters need to be treated separately. Thus the first approach of extending SMC introduces a great deal more complexity, and limits the choice of SMC techniques to just

a few. The latter, as used in the works of [14, 16, 110], raises an interesting privacy question: *How do two parties agree on a mapping without revealing information about their inputs?*

In this chapter, we present the first secure scaling protocol for two parties. It jointly enables them to agree on a mapping (by scaling) in a privacy-preserving manner and to use the scaling without the mapping being shared. The scaling factor is an intermediate value that can be hidden. The key building block for this protocol is a novel algorithm for privacy-preserving random number generation. We also show an efficient implementation of the protocol.

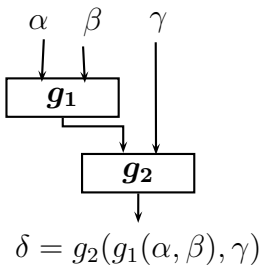
4.2 Secure Multiparty Computation

Secure Multiparty Computation (SMC) protocols enable parties to carry out distributed computation tasks without revealing their inputs to each other.

We describe different SMC techniques in Chapter 2. There are a variety of protocols for secure multiparty computation; these vary in assumptions, security guaranties, number of supported parties, performance and supported operations (see [58] for an overview). However, our protocol translates naturally into a Boolean circuit and we therefore use Yao's [144] protocol for SMC. We describe his protocol in detail in Section 2.3.3.

In the next paragraph, we will recap the definition of a Boolean circuit as both a refresher, and to clarify notation.

Boolean Circuit: A Boolean circuit consists of wires and gates. The wires transmit a value $\{0, 1\}$, and the gates compute a Boolean function on their input wires; they then output the result to another wire. This wire may then be connected to the input of another gate or be an output value of the circuit (Figure 4.2.1). Mathematically we describe a circuit by a series of functions $g_i(\alpha, \beta)$, $\alpha, \beta \in \{0, 1\}$, $g_i : \{0, 1\}^n \rightarrow \{0, 1\}$.



with $\alpha, \beta, \gamma, \delta \in \{0, 1\}$
and $g_i : \{0, 1\}^2 \mapsto \{0, 1\}$

Figure 4.2.1: A Boolean circuit consisting of 2 two-input gates

Once the input wires to a gate are given values α, β, it is possible to compute $g_1(\alpha, \beta)$ and assign it to the output wire which becomes an input to $g_2(\cdot, \cdot)$, etc. The output of the circuit is given by the values of the output wires of the circuit. Thus, computing the circuit C is essentially just allocating appropriate Boolean values to all wires of the circuit.

4.3 Secure Mapping

In order to apply a generic SMC protocol to real numbers, or floating point approximations, one can define a mapping from the real inputs into the integer space, and thereafter use conventional SMC.

The obvious approach to map real numbers to a finite set of integers is scaling and quantisation. Let $r \in \mathbb{R}$ be the real number input. Then $i = \lceil s \cdot r \rfloor$ is the mapping from r to i, where $\lceil \cdot \rfloor$ is the function that rounds to the nearest integer, and s is a scaling factor the parties agree on.

For example, let $A = (0.3, 7, 0.29)$ be the secret inputs of party A, and $B = (9, 3.3, 3.2)$ the inputs of party B, respectively. A scaling factor of $s = 100$ maps their inputs to $\tilde{A} = (30, 700, 29)$ and $\tilde{B} = (900, 330, 320)$.

It is easy to see that the scaling factor leaks information about the inputs, since it has to be chosen such that both all inputs are mapped into the finite set of integers and they are still distinguishable. Each party can support a different set of scaling factors, depending on their respective inputs.

If party A and B want to engage in a SMC protocol that supports 10 bit integer inputs (0 to 1023), then the set S_A of possible scaling factors s_A for the inputs of party A in this example is $S_A = \{s_A, \text{with } 100 \leq s_A \leq 146\}$ and $S_B = \{s_B, \text{with } 10 \leq s_B \leq 113\}$, respectively.

Revealing these sets to each other leaks information. In particular, the smallest and the biggest input can be approximated by dividing the integer input bandwidth by the biggest and smallest possible scaling factor. Both parties want to agree on a scaling factor without having to reveal any information about their supported sets other than that they contain the chosen scaling factor.

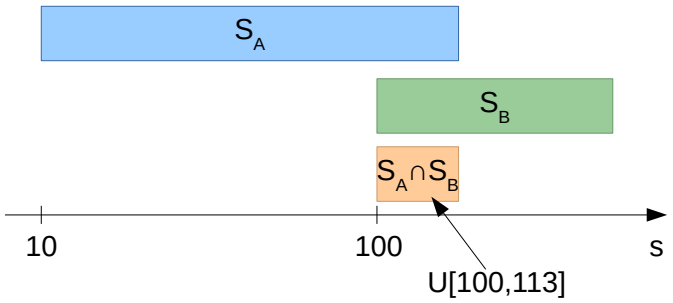


Figure 4.3.2: S_A and S_B show the sets of possible scaling factors for party A and B. We pick a factor uniformly at random out of the intersection $S_A \cap S_B$.

We propose two protocols which offer a trade-off between privacy and efficiency.

- The *Secure Scaling* protocol picks a scaling factor uniformly at random from the intersection of the ranges given by two parties. At the end of the protocol the scaling factor gets revealed to both parties. It offers the best efficiency, as only the determination of the scaling factor is done in the encrypted space. It also offers more flexibility, because after the parties mapped their respective inputs into the integer space, they can then engage in a secure multiparty computation choosing any SMC technique.

- However, revealing the scaling factor to the parties raises a privacy issue: it can be used to infer some information, as discussed in Section 4.4.4. To mitigate this issue, we propose a *Secure Mapping* protocol with better privacy guarantees at the cost of efficiency. It performs the whole mapping process in the encrypted space, and, by subsequently feeding the encrypted, mapped inputs into succeeding SMC protocols, the scaling factor stays hidden and cannot be inferred. Performing the mapping in an encrypted fashion introduces additional costs, and, as the outputs cannot be decrypted, the choice of privacy-preserving techniques for the subsequent protocol is limited. The best choice is to stay with the garbled circuit technique, however, there is a protocol [89] to convert garbled values into homomorphically encrypted values.

We first present the *Secure Scaling* protocol, as it is also the foundation of the *Secure Mapping* protocol. The basic idea is to first compute a secure set intersection and then, without revealing the intersection, pick an element at random (see Section 4.4.4). While secure set intersection protocols are readily available, we propose the first protocol (of which we are aware), to draw a random number from a private range.

4.4 Drawing Random Numbers from a Private Range

We do not want to reveal the range of possible scaling factors, once it is computed with the privacy-preserving set-intersection protocol. Instead, we keep it in the encrypted space and use it as the input to the random number algorithm. The goal of this algorithm is to pick an element uniformly at random out of the range, without giving the participants any more information than the randomly drawn element itself.

We first show the simple case where the range starts with 0 and has a

number of elements that is a power of two. We then allow for an arbitrary number of elements, still starting with 0, and finally we present the algorithm where both bounds of the range are arbitrary values.

4.4.1 Range $N_{2^m-1} = \{0, 1, 2, \ldots, 2^m - 1\}$

The set $N_{2^m-1} = \{0, 1, 2, \ldots, 2^m - 1\}$ is the set of integers that can be represented by an m-bit number. If we choose m random bits, each with probability $1/2$, the binary number denoted by these bits will be uniformly distributed over N_{2^m-1}. The algorithm combines random bits chosen by both parties, and then chooses m of these.

Let the private input m come from a finite set $I = \{0, 1, \ldots, n\}$, which is agreed on by both parties, and, $N_{2^n-1} = \{0, 1, \ldots, 2^n - 1\}$ be the set of all n-bit integers. In this context m would be the output of a preceding SMC protocol. Both parties choose $r^{(A)}, r^{(B)} \in_R N_{2^n-1}$, respectively, where $\in_R$ means chosen uniformly at random from the set. The algorithm first combines the random n-bit inputs by the bitwise exclusive OR operation (XOR) to get r, and then selects the m least significant bits of r by computing the output $x = r \bmod 2^m$.

Algorithm 3 *urandom1:* Drawing x randomly from $\{0, 1, 2, \ldots, 2^m - 1\}$

Input: (private) $m \in I$

Output: $x = \text{urandom1}(\{0, 1, \ldots, 2^m - 1\})$

$r \leftarrow r^{(A)} \,\text{XOR}\, r^{(B)}$ $\{r^{(A)}, r^{(B)} \in_R N_{2^n-1}$, where $r^{(i)}$ is provided by party $i\}$

$x \leftarrow r \bmod 2^m$

return x

Correctness: It is easy to see that $x \in \{0, 1, \ldots, 2^m - 1\}$ since that is exactly the co-domain of $r \bmod 2^m$. We also have to show that x is a uniformly distributed random variable over that range.

Lemma. *If at least one of $r^{(A)}$ and $r^{(B)}$ is chosen uniformly at random out of $N_{2^n-1} = \{0, 1, \ldots, 2^n - 1\}$ then $r = r^{(A)} \,\mathrm{XOR}\, r^{(B)}$ is a random number uniformly distributed over N_{2^n-1}.*

Proof. Let $N_{2^n-1} = \{0, 1, 2, \ldots, 2^n - 1\}$ be the set of all integers that can be represented by n bits. If $r^{(A)}$ is a random variable on N_{2^n-1}, then there exists a unique random vector $(r_1^{(A)}, \ldots, r_n^{(A)})$ on $\{0,1\}^n$ such that $r^{(A)} = \sum_{i=1}^{n} 2^{i-1} r_i^{(A)}$. If $r^{(A)}$ is uniformly distributed over N_{2^n-1} then the $r_i^{(A)}$'s are mutually independent Bernoulli random variables with parameter $1/2$. $r = r^{(A)} \,\mathrm{XOR}\, r^{(B)}$ can now be written as $r = \sum_{i=1}^{n} 2^{i-1} r_i$ with $r_i = r_i^{(A)} \,\mathrm{XOR}\, r_i^{(B)}$. Note that the XOR operation returns 1 iff both arguments are different.

Assume that $r^{(A)}$ is uniformly distributed. Therefore

$$\Pr[r_i = 1 | r_i^{(B)} = 0] = \Pr[r_i^{(A)} = 1] = 1/2$$
$$\Pr[r_i = 1 | r_i^{(B)} = 1] = \Pr[r_i^{(A)} = 0] = 1/2.$$

Note that the value of $r_i^{(B)}$ has no influence on $\Pr[r_i = 1]$. Thus $\Pr[r_i = 1] = \Pr[r_i = 0] = 1/2$. XOR is a bitwise operation and the r_i are mutually independent and thus r is uniformly distributed over N_{2^n-1}. □

Now $x = r \bmod 2^m$ can be rewritten as $x = \sum_{i=1}^{m} 2^{i-1} r_i$ since the mod 2^m operation selects the m least significant bits of r. The r_i are mutually independent Bernoulli random variables with parameter $1/2$, hence x is a random variable uniformly distributed over $\{0, 1, \ldots, 2^m - 1\}$.

Security: We want to keep the input m private. We ensure that the parties do not learn it via use of the garbled circuit technique (see Section 4.5). What remains evident is that neither party can choose their input to manipulate the output. A successful attack would distort the uniform distribution of the output. However, we know from Lemma 4.4.1 that the output is uniformly distributed as long as at least one input is uniformly distributed. So even if party A (or party B) deviates from the protocol, and

deliberately chooses a specific value for $r^{(A)}$ (or $r^{(B)}$), the output will remain uniformly distributed.

4.4.2 Range $N_q = \{0, 1, \ldots, q\}$

In this section, we relax the restriction that the size of the range must be an exact power of two. Now, we allow any range $N_q = \{0, 1, \ldots, q\}$ with $q \in \mathbb{N}$. The number of elements in that range is not necessarily a power of two and therefore we can't directly apply Algorithm 3.

We use the acceptance-rejection method to construct a Las Vegas type algorithm[1] that uses Algorithm 3 repeatedly until it produces a value in the required range. To do this, we extend N_q to N_{2^m-1} so that it is of the form of Algorithm 3. That is, we choose the unique $m \in \mathbb{N}$ with $2^{m-1} - 1 < q \leq 2^m - 1$, and then run the algorithm as described in Algorithm 4. This approach translates naturally into a compact circuit with a number of gates that is linear in the input size.

Algorithm 4 *urandom2:* Drawing x randomly from $\{0, 1, \ldots, q\}$

Input: (private) $q \in N_{2^n-1}$

Output: $x = \text{urandom2}(\{0, 1, \ldots, q\})$

$m \leftarrow \lfloor log_2(q) + 1 \rfloor$

repeat

$x \leftarrow \text{urandom1}(\{0, 1, \ldots, 2^m - 1\})$

until $x \leq q$

return x

---

[1] A Las Vegas algorithm is a randomised algorithm that "gambles" with the resources used for the computation. That is, its runtime is random, but the *expected* runtime is finite. In contrast, a Monte Carlo algorithm is a randomised algorithm whose runtime is deterministic, but whose output may be incorrect with a certain probability. Thus it "gambles" with the correctness of the result.

Correctness: When the algorithm terminates $x \in_R \{0, 1, \ldots, q\}$ since the exit condition ensures $x \leq q$, and *urandom1* produces non-negative numbers. x is uniformly distributed since acceptance-rejection sampling of a subset of a uniform distribution is again uniformly distributed [139].

The number of iterations of the loop follows a geometric distribution. Let X be a random variable describing how many iterations Algorithm 3 takes to get a valid result. The probability that $X \leq k$ with $k \in \mathbb{N}$ is

$$\Pr[X \leq k] = 1 - (1 - \Pr[x \leq q])^k.$$

The probability that the exit condition is fulfilled in one iteration is

$$\Pr[x \leq q] \geq \frac{2^{m-1} + 1}{2^m} > \frac{1}{2},$$

because $2^{m-1} - 1 < q \leq 2^m - 1$, and so $\Pr[X \leq k] > 1 - (1/2)^k$.

That means that even in the worst case the expected number of iterations is less than 2, and the probability of less than 10 iterations is greater than 99.9%. We illustrate the performance in Section 4.7.

Security: Again, neither party can distort the uniform distribution of the random value by the same argument as for Algorithm 3. Nothing is learned from the number of calls to *urandom1* because it is purely probabilistic.

4.4.3 Range $N_{p,q} = \{p, p + 1, \ldots, q\}$

In the most general case where the range is arbitrary, we first shift it to zero, then use Algorithm 4 to compute a random value, and finally shift it back to the initial range (See Algorithm 5 for details).

Correctness and Security: Correctness and security follow from the same arguments as for Algorithm 4.

Algorithm 5 *urandom3:* Drawing x randomly from $\{p, p+1, \ldots, q\}$

Input: (private) $p, q \in N_{2^n}$

Output: $x = \text{urandom3}(\{p, p+1, \ldots, q\})$

$m \leftarrow \lfloor \log_2(q-p) + 1 \rfloor$

repeat

$s \leftarrow \text{urandom1}(\{0, 1, \ldots, 2^m - 1\})$

until $s \leq q - p$

return $x = s + p$

4.4.4 Secure Scaling

Once we have a random number generator, we can build an efficient solution for the secure scaling problem.

Both parties input their smallest $(p^{(A)}, p^{(B)})$ and biggest $(q^{(A)}, q^{(B)})$ possible scaling factors. The inputs p and q must be integer values, as we subsequently use them as inputs in a SMC protocol. If the parties require non-integer scaling factors, they can agree on a mapping between this non-integer space and the integers beforehand. The first step is to determine the intersection of these ranges by computing the boundaries of the intersection as $p = \max(p^{(A)}, p^{(B)})$ and $q = \min(q^{(A)}, q^{(B)})$. In the second step, we use the random number generator to select an element out of $\{p, p+1, \ldots, q\}$.

Algorithm 6 The Secure Scaling algorithm

Input: $p^{(A)}, q^{(A)}, p^{(B)}, q^{(B)} \in N_{2^n-1}$

Output: $s \in_R \{p^{(A)}, p^{(A)}+1, \ldots, q^{(A)}\} \cap \{p^{(B)}, p^{(B)}+1, \ldots, q^{(B)}\}$

$p \leftarrow \max(p^{(A)}, p^{(B)})$

$q \leftarrow \min(q^{(A)}, q^{(B)})$

$s \leftarrow \text{urandom3}(\{p, p+1, \ldots, q\})$

return s

Correctness and Security: Correctness and security follow from the same arguments as for Algorithm 4. However, although the SMC technique guarantees that nothing can be learned about the inputs apart from what can be inferred from the output, revealing the chosen scaling factor introduces a privacy issue. Let $p^{(A)} = 1$ and $q^{(A)} = 100$. If the algorithm outputs $s = 100$ then party A can infer that party B's smallest input $p^{(B)}$ is likely to be close to s. Assume that $p^{(B)} = 100$, then the intersection of the two sets contains only one element and therefore the algorithm outputs $s = 100$ with a probability of 100%. In contrast, if $p^{(B)} = 1$, then the intersection contains 100 elements, and the probability that the algorithm outputs $s = 100$ is only 1%. This privacy issue has to be considered when using the secure scaling protocol. For the case where it is deemed unacceptable, we also offer the *Secure Mapping* protocol as described in Section 4.6.

In the following section, we show how to implement all of the steps needed in the Secure Scaling algorithm using garbled circuits.

4.5 Secure Scaling with Boolean Circuits

We compute the secure scaling algorithm with Yao's garbled circuit technique by expressing it as a Boolean circuit. Boolean circuits are easily combined, so we show the subcircuits corresponding to the elementary operations in the algorithm.

We describe the complexity of each subcircuit by the number of non-linear two-input gates in relation to the number of bits n needed to represent the inputs $p^{(A)}, p^{(B)}, q^{(A)}, q^{(B)}$. A linear input gate has an even number of zeros and ones in the truth table. The linear gates for a constant output or the (negated) identity of an input wire can be trivially optimised away, e.g. XOR gates can be evaluated essentially for free [90], therefore the dominating factor for efficiency of the circuits is the number of non-linear gates.

- $(\min(q^{(A)}, q^{(B)}),\ \max(p^{(A)}, p^{(B)}))$: To compute these we use the inte-

ger comparison circuit described by Kolesnikov *et al.* [88], it has a complexity of n non-linear gates.

- $(m \leftarrow \lfloor \log_2(q-p)+1 \rfloor)$: In this step we don't actually have to compute m, because all we need later on is a bit mask to select the $\lfloor \log_2(q-p)+1 \rfloor$ least significant bits. Therefore we first compute $t = q - p$ with the integer subtraction circuit described in [88] and then we use a chain of OR-gates (see Figure 4.5.3) to calculate the mask $2^{\lfloor (\log_2(t)+1 \rfloor} - 1$. This circuit consists of $n - 1$ non-linear gates.

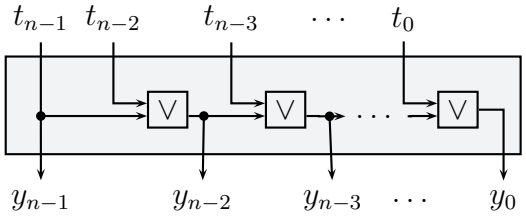


Figure 4.5.3: A chain of OR gates to compute the mask $y = 2^{\lfloor \log_2(t)+1 \rfloor} - 1$.

- $(r = r^{(A)} \,\mathrm{XOR}\, r^{(B)})$: r is just a bitwise XOR between $r^{(A)}$ and $r^{(B)}$. Therefore the complexity is 0 non-linear gates.

- $(s = r \bmod 2^m)$: Computing modulo a power of two is the special case where we just want to select the m least significant bits of r. We can achieve this by computing a bitwise AND between r and $2^m - 1$, the bit-mask with the m least significant bits set to 1. This is exactly the bit-mask we computed before. The complexity is n non-linear gates.

- (repeat until $s \leq q-p$): Note that this loop has an unknown number of iterations, and therefore it is impossible to generate the whole circuit to compute the loop beforehand. However, in this case, where the exit condition of the loop does not reveal any sensitive information, we can use a step-by-step approach. We make use of the feature of dynamic circuit generation of the SFE-framework described in Chapter 3. That is, the creator generates the circuit for one round of the

loop, and then the evaluator evaluates the circuit, thus revealing the result of the exit condition. Depending on that result, the creator then generates either another round of the loop, or continues with the remainder of the algorithm. Note that the disclosure of the result of the exit condition gives neither party an advantage in inferring the other party's input, so long as their random inputs are kept private. This privacy is guaranteed by the garbled circuit technique.

For the exit condition, we can reuse $q - p$, which we computed before. Thus we only need an integer comparison circuit [88], which has a complexity of n non-linear gates.

- $(x = s + p)$: We use the addition circuit of [88] to compute this sum. Again, the complexity is n non-linear gates.

Overall complexity: Let X describe the number of iterations of the repeat-until loop in Algorithm 4. Then the number of non-linear gates add up to $2n + 2n - 1 + X(2n) + n = 5n - 1 + 2nX$. Since X follows a geometric distribution with success probability $1/2 < p \leq 1$, we know that $1 \leq \mathbb{E}[X] < 2$, thus the expected overall complexity is less than $9n$.

4.5.1 Implementation

We chose the ME-SFE framework of Chapter 3 to implement our example of the random scaling factor. Amongst other optimisation techniques used in this framework, the following are particularly useful for our application:

- Pipelined circuit execution: The circuit generation and evaluation processes are overlapped in time [78], thereby removing the need to construct the complete circuit before the evaluation; this is useful here because we cannot build the circuit in advance, given that the number of iterations is dynamic.

- Oblivious-Transfer extension: In [79], Ishai *et al.* show how to efficiently extend Oblivious Transfer. One first must execute a certain amount of conventional OTs, and then (by using this result) one can generate a virtually unlimited number of very efficient OTs (see also Section 3.3.1). Using the OT extension technique provides a speedup by orders of magnitude; indeed, whereas the execution time of a traditional OT is somewhere in the order of milliseconds, with OT extensions it reduced to mere microseconds only (see benchmark in Section 3.6.1).

The ME-SFE framework contains a library of circuits for common arithmetic, which can be easily combined to describe the desired function. One can combine circuits from, (and add circuits to), the library by extending the *CompositeCircuit* class. For example, the implementation of the chain of OR-gates circuit (as shown in Figure 4.5.3) is carried out by defining subcircuits, and connecting them with wires as shown in Figure 4.5.4.

The source code of the implementation can be found as part of the ME-SFE package available for download at: `http://code.google.com/p/me-sfe/`.

4.6 Secure Mapping

Revealing the scaling factor can lead to privacy issues, as described in Section 4.4.4. One way to mitigate this issue is to carry out the full mapping in the encrypted space.

The most common way to represent a real number input is by floating point approximation. A floating point approximation of a number r consists of a significand s, a base b, and an exponent e, such that $r = s \times b^e$. Mapping r to an Integer is a three stage process.

1. First we multiply the significand with the scaling factor.

```
public class NextBitMask extends CompositeCircuit {
  protected void createSubCircuits() throws Exception {
    for(int i=0; i<l-1; i++){
      subCircuits[i] = OR_2_1.newInstance();
    }
    super.createSubCircuits();
  }
  protected void connectWires() throws Exception {
    for(int i=0; i<l-1; i++){
      inputWires[i].connectTo(subCircuits[i].inputWires,0);
    }
    inputWires[l-1].connectTo(subCircuits[l-2].inputWires,1);
    for(int i=0; i<l-2; i++){
      subCircuits[i+1].outputWires[0].connectTo(
                                    subCircuits[i].inputWires,1);
    }
  }
  protected void defineOutputWires() {
    for(int i=0; i<l-1; i++){
      outputWires[i] = subCircuits[i].outputWires[0];
    }
    outputWires[l-1] = inputWires[l-1];
  }
```

Figure 4.5.4: Implementation of the chain of OR-gates circuit as shown in Figure 4.5.3

2. Then we use the information of the exponent to select the integer part of that product.

3. Finally, we have to account for rounding by checking the first digit after the radix point, and, if closer to the next integer, increment the result accordingly.

Note about privacy: Obviously, the mapped inputs cannot be revealed to either party, as this would allow them to compute the scaling factor, leading to the privacy issues discussed in Section 4.4.4. Therefore, the mapped inputs must be kept in the encrypted space and the subsequent protocol must be chosen such that the output does not reveal any hints to infer either the scaling factor, or any of the inputs. Although we require that the mapped inputs are kept in the encrypted space, that does not mean that the subsequent protocol is limited to the garbled circuit technique, as there is a efficient protocol [89] to convert garbled values into homomorphically encrypted values.

4.6.1 Implementation

We will only sketch the implementation of the secure mapping protocol.

- For the first part, we use the multiplication circuit of Section 3.5. It has a complexity of $\mathcal{O}(l^{\log_2 3}) \approx \mathcal{O}(l^{1.585})$.

- Every output digit can be either zero or any of the input digits. Therefore we need for every output digit $l+1$ multiplexer to select the correct digit. In total, this stage has a complexity of $\mathcal{O}(l^2)$.

- For the last stage we need $l + 1$ multiplexer to select the first digit after the radix point ($\mathcal{O}(l)$). This is then fed into a comparison circuit of [88] ($\mathcal{O}(l)$) and the result added to the integer with an addition circuit [88] ($\mathcal{O}(l)$).

4.7 Measurements

All our measurements were run on an iMac with a Core i3 3Ghz processor, running Mac OS X 10.6.8 and Java 1.6.0_31. We ran measurements for four different input sizes (10, 100, 1000 and 10000 bits). For each size we ran the secure scaling algorithm 10000 times with inputs $(p^{(A)}, p^{(B)}, q^{(A)}, q^{(B)})$ generated uniformly at random from the set of non-negative integers able to be represented by the given number of bits. Figure 4.7.5 shows the distributions of the runtimes for the different input sizes. The single red line denotes the median, the blue box include the data points from the 25^{th} to the 75^{th} percentile, and the whiskers include all points up to 1.5 times the size of the blue box. The resolution of the measurements is 1 ms; therefore, the data points for the 10 bit input size are not particularly precise, and are only included in the graph to highlight the overall trend (median, 25^{th}, 75^{th} percentile and whiskers are all at 1 ms). The linear circuit complexity with respect to input bit length is clear. Note also the very strong right skewness of the data.

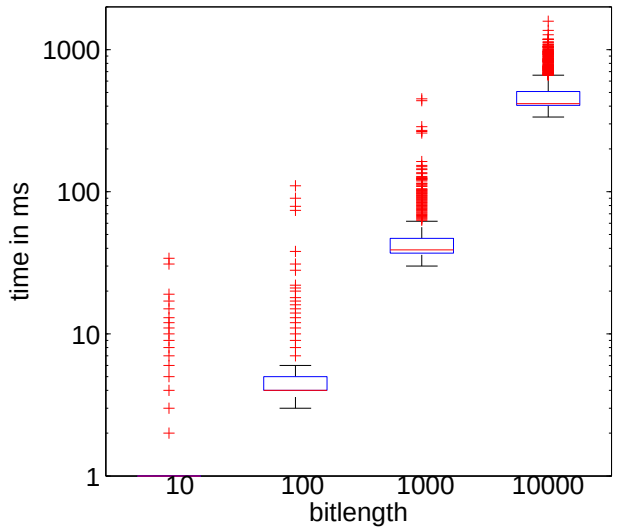


Figure 4.7.5: Runtime distributions of the secure scaling algorithm for different input sizes.

Figure 4.7.6 shows the complementary cumulative distribution functions of the number of iterations for different input sizes (that is, the probability that a run has more than X iterations). We also added the worst case scenario for 1000 bit inputs, whereby the inputs are chosen such that the private range is 2^{999} and therefore the probability that the exit condition of the loop is fulfilled is $(2^{999}+1)/2^{1000} \approx 1/2$. We see that the input size has little effect on the distribution. Even for the worst case, the probability for a high number of iterations drops rapidly.

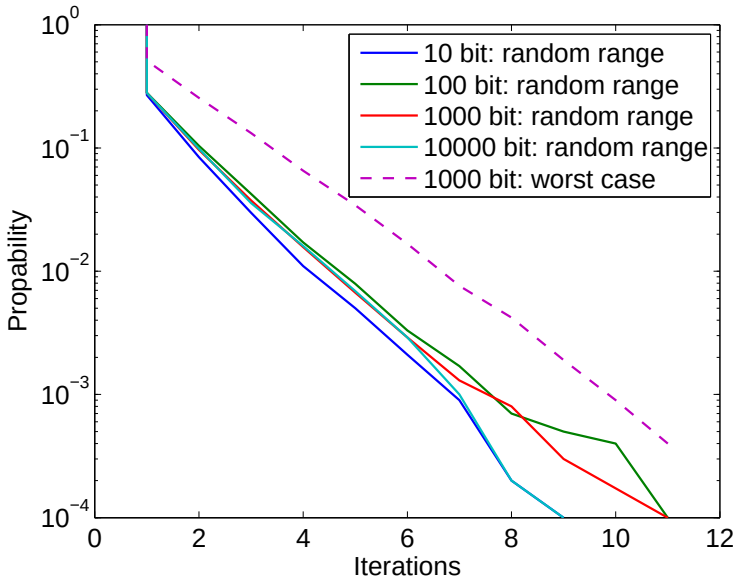


Figure 4.7.6: Complementary cumulative distribution functions of the number of iterations for different input sizes.

4.8 Conclusions

In this chapter we have presented a protocol to solve the secure mapping by scaling problem. It enables two parties to convert a real-numbered privacy-preserving problem into the integer domain. Previous solutions to this problem either limit the choice of SMC techniques, or use a public, pre-defined scaling factor. The first approach introduces more complexity, as the SMC

operations on non-integer values are more computationally expensive than their integer counterparts. Our solution does not introduce any limitations on the choice of SMC techniques. The latter approach is not applicable in a general sense, as if the input domains for the parties involved differ then negotiating a scaling factor most likely leaks information about the inputs. Our protocol enables the parties to input their respective range of possible scaling factors without revealing this information to their counterpart. The main component of our solution is, to our knowledge, the first privacy-preserving random number generator. We believe that it may be a useful component for other privacy-preserving protocols. We show the practicality of our scaling solution by an implementation of the protocol, using the ideally-suited ME-SFE framework presented in Chapter 3.

This chapter concludes the part of this thesis where we improve the practicability of SMC. In the following chapters we focus on applying SMC techniques to problems of network management. In the next chapter we present a routing protocol which preserves the privacy of the routing configuration.

Chapter 5

Privacy-Preserving Vector-Based Routing

After working on the practicability of secure multiparty computation in the previous chapters, we now focus on applications of SMC to problems of network management. In this chapter we present a routing protocol that preserves the privacy of the routing configuration.

Security of routing protocols is a critical issue, as shown by the increasing number of attacks on the Internet's routing infrastructure [49]. One often overlooked aspect of security is privacy. In the context of a routing protocol, this essentially means the ability of a router to keep information (such as its routing policies) private. BGP does this to some extent through design. An Autonomous System's policies are not explicitly revealed to other participants in the routing protocol. Nevertheless, BGP still reveals a great deal of information about the Internet and its participants. We propose a privacy-preserving routing protocol, called STRIP, that reveals very little information to participants in the protocol. For instance, participants can find shortest-paths to destinations in the network without ever learning the path lengths. Such privacy could be useful for a range of reasons; these include preserving the proprietary information captured in a routing policy, or preventing an attacker from gaining valuable information about the network.

We show the feasibility, performance, and costs of STRIP, with simulations and implementations of the protocol.

5.1 Introduction

There is a long list of desirable features for a routing protocol. For instance, it should be robust, distributed, scalable, and easy to configure. There are now many protocols with different sets of these properties, but more recently security of routing protocols has become a major issue. The reason for this lies in the spread of routing protocols between untrusted parties. The canonical example is inter-domain routing. The defacto standard inter-domain routing protocol is the Border Gateway Protocol (BGP). The rapid expansion of the Internet has led to proliferation of BGP speaking Autonomous Systems (ASes): more than 40,000 at the time of writing. In the Internet of yesteryear, the BGP speaking networks were almost like a club; membership assured an element of civility. However, in recent years, such trust has led to problems. There have been many documented problems with BGP: spammers have exploited security vulnerabilities of BGP to send unwanted, or illegal emails [123], and accidental and/or deliberate hijacking of address space has caused large scale disruptions (for just a few examples see [42, 73, 102]). Most of the resulting work on adding security to BGP has focussed on authentication.

One aspect of securing routing protocols that has not received widespread study is privacy. Of course, one could encrypt the individual transactions of a routing protocol to prevent eavesdropping by external assailants. However, privacy with respect to the other participants in a routing protocol is a much more interesting problem. Routing protocols are generally designed to spread information. This is seen as a necessary step, as distributed control is one of the philosophical grounding points for Internet design; yet nonetheless there are good reasons to wish to preserve privacy

of routing information. The routing information of each party may contain proprietary information that could be commercially or politically sensitive, or may contain information rendering an attack against a relevant party far easier.

BGP implicitly acknowledges the dichotomy of information hiding/spreading. The protocol allows for each AS to have flexible, heterogeneous and dynamic policies, and BGP attempts to jointly determine a solution to these routing policies. However, BGP hides internal policies by performing a "best route" computation locally, and passing on the result. It doesn't pass on the policies used to make the decision. BGP's route computation passes only simple data (such as AS-paths) and locally defined attributes (such as MEDs and communities). This has resulted in a protocol that is not transparent. Its behaviour is unpredictable, and exhibits slow convergence [93], persistent oscillation [68,138], and other negative features [30,67]; nevertheless, ISPs have clearly been willing to trade-off such negatives against their desire to preserve privacy (BGPv4 has been a defacto standard for nearly two decades [125]).

But does BGP hide enough? BGP routing data has been used to build AS-level topologies of the Internet, and to infer relationships between ASes [105,136,140,142]. The same ideas can and have been used for tasks such as inferring customers of an ISP. This type of business intelligence can provide a competitor with an unfair advantage. A further, more sinister, use for such intelligence arises when one considers an antagonist planning an attack. The more information about a network an antagonist can gather, the more likely its attack will be successful. It is precisely this sort of information that would be needed to mount a malicious hijack of address space [7,23,118].

Although attacks on BGP in the Internet are on the rise [49], the majority of participants in BGP are well intentioned, and some problems (in the routing infrastructure) appear to have been caused by mistakes rather than deliberate malfeasance. There are other networks where security is the

paramount concern, rather than something added on after the fact. There are network operators who may wish to co-operate from time-to-time (i.e., by sharing traffic), yet who remain concerned that private details do not leak from one network into another. A simple example exists in the military setting, where national armed forces often co-operate in joint missions – where lives depend upon being able to share accurate, up to date information. However, today's allies may be tomorrow's combatants, and so parties may simultaneously wish to maintain secrecy regarding their networks' design and capabilities. Furthermore, in the operations of their own networks, they may wish to limit the damage that can be done if a router (or group of routers) is compromised. It is therefore interesting to explore the limits to which a routing protocol can preserve the privacy of its participants.

Here, we shall thus present a routing protocol aimed at preserving exactly that – privacy. We show that a great deal of information can be hidden; in particular, we show that a common routing algorithm — the distributed Bellman-Ford shortest-paths algorithm — can be performed without participants learning distances!

We call our new protocol STRIP (Secure-Transitive RIP), after the iconic distance-vector protocol RIP (the Routing Information Protocol). STRIP conceals nearly every aspect of the network from its participants. Participants in the protocol learn their neighbours, and the next-hop router on the route to a destination, but they discover very little else. In particular, participants do *not* learn any routes or distances (other than to their immediate neighbours).

In addition, the algorithm has other desirable properties. It uses a strong method to authenticate potential paths, step by step, much as stronger versions of BGP security improvements aim to do. Thus we achieve authentication almost for free.

It is also easy to see how the algorithm can be extended. BGP uses a path-vector algorithm that resembles a distance-vector protocol in some

respects, and this chapter also presents a number of ways in which our protocol can start to be adapted to the path-vector setting.

As always there are costs to security. We have implemented the proposed protocol in both simulation software, and as a real distributed routing protocol; subsequent analysis of the performance overhead reveals that it is not trivial. We do not suggest that the costs are warranted for the Internet in general, but we believe that a thorough understanding of what is possible should inform future routing protocol designers, and we leave it to them to choose the tradeoff between privacy and overhead.

Privacy-preserving ideas have been applied to interdomain routing before. Zhao *et al.* [146] showed a verification mechanism for routing decisions, and Gupta *et al.* [70] use a *secure outsourcing* approach to move the routing decision process from the routers to a centralised computation cluster. Our work, which is motivated by an earlier presentation [129], maintains the distributed nature of a routing protocol, and we do not rely on any additional players except a key distribution mechanism (PKI) which now exists for BGP in RPKI [94]. Another difference is the type of information that is kept private. Gupta *et al.* do not consider network information private, whereas our solution not only protects policy but also a great deal of the network information.

5.2 Secure Multiparty Computation

Secure multiparty computation protocols are a set of techniques, often cryptographic in nature, which enable parties to carry out distributed computation tasks without having to reveal their private data.

We give a detailed description in Chapter 2. In our protocol we use homomorphic encryption to securely compute a distributed sum.

5.2.1 Distributed Sum with Homomorphic Encryption

An encryption scheme is called *homomorphic* if there exists an operation on two ciphertexts that is equivalent to another operation on the corresponding plaintexts: i.e.,

$$\mathrm{Enc}(x) \odot \mathrm{Enc}(y) = \mathrm{Enc}(x \oplus y),$$

for some operations $\odot$ and $\oplus$. In our protocol we use an *additive* homomorphic encryption scheme, where the operation $\oplus$ corresponds to standard arithmetic addition.

The elegant feature of this approach is that we can create a sum of a series of values held by different parties, but which is encrypted. Only the holder of the private key can decrypt it and determine the sum.

In detail, the secure distributed sum is computed as follows.

1. Assume we have n parties $P_1, P_2, \ldots, P_n$ with corresponding inputs $x_1, x_2, \ldots, x_n$, and we wish (at completion) for party P_n to know the sum
$$X = \sum_{i=1}^{n} x_i.$$

2. Each party encrypts their data with the public key of P_n to obtain $y_i = \mathrm{Enc}(x_i; P_n)$.

3. WLOG we assume that they transmit cumulative sums to the next party in sequence, i.e., party P_i sends the following to P_{i+1},
$$y_1 \odot y_2 \odot \cdots \odot y_i.$$

4. Finally, P_n receives $y_1 \odot y_2 \odot \cdots \odot y_{n-1}$, and decrypts it with its private key and adds its own value x_n.

Note that for more than two parties P_n learns nothing about $x_1, \ldots, x_{n-1}$ apart from the sum $\sum_{i=1}^{n-1} x_i$.

There are a number of possible homomorphic encryption schemes (see overview in Section 2.2). Here we use Paillier encryption [114], which is

homomorphic with respect to summation (as required), and comparatively simple to implement. We describe Paillier's scheme in Section 2.2.2.

5.2.2 Key distribution problem

The one critical requirement for the public-key encryption system is a Public-Key Infrastructure (PKI). For a router to determine a route to a destination, it has to know its public key. That is, it has to know the key before it has a valid route to the destination. As such, it cannot ask the destination for the key beforehand; yet even if this was possible, how can the router know that this public key does indeed belongs to the destination, and not to an attacker pretending to be the destination router?

This problem is not limited to our scenario – indeed, it is well known in public-key cryptography applications. The most prominent solution to this problem is the introduction of a trusted third party who provides the PKI. They act as a public-key broker: verifying and linking identities to public keys, and distributing those keys in a secure manner.

In general, creation of PKIs is non-trivial. However, the pressing need for improved BGP security means the problem has been tackled for Internet routing. A PKI system specially designed for routing infrastructure, called RPKI (Resource PKI) [94], has already undergone some testing [116].

5.3 STRIP

We call our protocol STRIP (Secure-Transitive Routing Information Protocol). Its aim is to find the shortest paths through a network, while revealing minimal information about that network.

The purpose of a routing protocol is to provide an automated and distributed means to create *routing tables* at each router. These are essentially tables of destinations accompanied by *next hops* (the first step from the current router along the path to the destination), along with some information

about the paths (such as distances).

The "destinations" in our protocol might be the routers themselves, but could equally be some aggregate such as the Autonomous Systems of BGP, or some set of subnets attached to routers. However, for the sake of simplicity, we shall equate destinations and routers here.

There are several approaches for calculating shortest-paths. *Path-Vector Protocols* (PVP) use an interesting approach that combines the information passing and decision processes, and it is this approach that we shall generalise in STRIP.

PVPs are sometimes called "routing by rumour". In a PVP, each router shares its routing table entries by announcing them to its immediate neighbours. Thus the neighbours learn of potential destinations, and potential paths towards these destinations. The advertised information in the table includes the path (and in the case of BGP it contains other metrics), and these can be used to discriminate between potential choices when a router learns of more than one path.

In BGP, the best-path decision is made based on multiple metrics; however, here we will concentrate on shortest-paths, thereby avoiding some of the intractable problems of BGP convergence [68,138]. Nonetheless, it should be clear that the approach we propose generalises to allow for multiple metrics composed as lexicographic products [115].

In our protocols we allow each link in the network to be assigned a weight, and the weight of a path is the sum of its constituent link weights. The selection criterion then is to select the path with the lowest weight (the shortest path).

The announcements of a typical PVP contain the destination, the weight of the announced path to the destination, and a list of routers contained in the path. Thus selecting a path is just a matter of comparing the new path weight with the existing one, and loops can be detected (and eliminated) if the receiving router is already present in the list of routers.

Clearly, such announcements leak information about the configuration of the network. As many of these are sent, and received, they can be used to create a combined picture of the network and its policies [105]. In contrast, the proposed STRIP protocol keeps this information secret.

However, there is still a minimal amount of information that must be public. We assume that routers know their neighbours (this is reasonable as establishing a link requires co-operation). We also assume that a router knows (or determines) the link weight of the directed edges from it, to its neighbours (the weights don't need to be symmetric, and the routers don't need to know the distances of the links towards them). In practice though, weights might be determined cooperatively, so as to be symmetric. Finally, we assume that all links in the network are bidirectional, because messages must be able to travel in both directions along a path to enable the shortest path computation in our protocol.

5.3.1 Route propagation

We make two major modifications to a typical PVP. The first is that we alter the information routers transmit from their routing tables. They still transmit potential destinations, but these announcements are now simply a list of destinations without any information about the path. A router will then learn of alternative next-hop routers towards a destination.

The second major difference is that the router does not itself decide between these alternatives. In order to decide between them, it starts a *shortest path computation* (SPC), in which the destination router is asked to make the decision (described in more detail in Section 5.3.2).

Although we modify the PVP significantly, we want to stress that we did not change the underlying mechanism for route propagation: all known routes to a destination are assessed, and the shortest is selected and announced to the neighbours. Hence, its convergence properties are almost identical to those of conventional PVPs.

A schematic overview of the operation of STRIP is given in Figure 5.3.1. It shows the process: router C receives announcements of the destination D from A and B; C sends a shortest-path computation request to D via the two alternative paths; D makes the computation, and responds to C with the best path; and then (as in all PVPs) C would announce its new destination to its neighbours, who might then commence their own computation.

It may seem, superficially, that D needs to know about C before it can return the message; but we will explain in Section 5.3.2 how to avoid this problem. As such, although the process involves more message passing, it is logically identical to the standard PVPs, which are known to converge correctly for shortest paths (given non-negative weights).

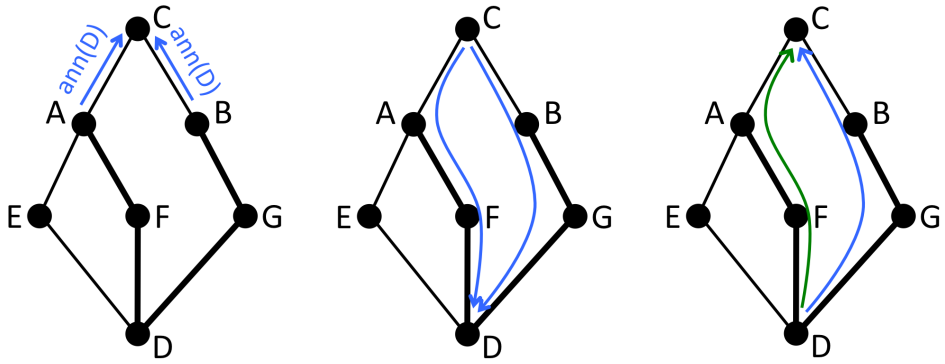
5.3.2 Shortest Path Computation (SPC):

The main change to the PVP is the way shortest paths are selected. In STRIP, a router starts a SPC: a distributed computation involving the routers on the known paths to the destination. The idea is that the originating router sends a "probe" message to the destination along all paths known to it. The intermediate routers add the path weight for the respective links to the messages; and lastly, the destination router, after receiving all probes, decides on the shortest path and sends a reply back to the originating router.

We could easily do this with all messages in the clear, resulting in a modified PVP. This would have advantages in itself. For instance, the response proves that the path is valid: this is not at all guaranteed by the current version of BGP (hence the need for BGPsec).

However, we make the additional change that the weight-sum is computed using homomorphic encryption, as described in Section 5.2.1, and we anonymise so that D learns little from its role in the computation.

In detail: the originator of the SPC sends an SPC-request to all neighbouring routers that have announced a route to the destination. An SPC-request contains:



(a) Each node that knows of a route to a destination advertises it to its neighbours. In this example A and B announce the existence of a path to D to router C.

(b) C starts a shortest-path computation by sending requests to the announcing routers A and B, who forward the request on their known routes to the destination D. Along the path each involved router adds the distance to the next hop to the path's distance stored in the request.

(c) After receiving all requests for this SPC, D determines the shortest path, and then sends replies to C along the reverse paths. C then updates its forwarding table and sends an announcement of the newly learned route to D (in green) to its neighbours.

Figure 5.3.1: Operation of STRIP. The dark lines show the pre-existing paths (though note that each router only knows its next hop).

- The address of the destination.
- A random computation ID, unique to each SPC.
- A random path ID, unique to each path in the computation.
- A distance field, holding the encrypted sum of the weights of the links of the path.
- A random "originator" encryption key for a symmetric encryption scheme, encrypted with the destination router's public key.
- A timestamp, holding the creation time of the request.
- The distance of the current shortest path to the destination known to the originator, encrypted with the destination router's public key.

Note that only the destination router can decrypt the message details. No one else can learn anything about the distance, and although D learns a set of distances, it does not know the origin, since the random path and computation IDs and the random key are anonymised.

If an intermediate router receives a SPC request, it looks up the next hop to the destination in its routing table (it must have one for this to be a valid path), and adds the distance for that link to the encrypted distance field. It then stores the last hop, path ID and computation ID in a temporary routing table, and sends the request to the next hop. The temporary routing tables serve the purpose of enabling the network to route the responses back on the same path as the corresponding request, without revealing the identity of the originator. This is the reason why the protocol requires bidirectional links. Otherwise a response might never reach its destination.

If the destination router receives a SPC request, it waits a certain period of time (the `waitForRequests` time) for other requests of the same computation to arrive. Any requests with the same computation ID after the timeout has expired are ignored. It then decrypts all distances, chooses the

smallest one and prepares the response messages. A response is generated for every request it has received. The response contains the path ID, computation ID, and the ID of the path with the shortest weight, encrypted with the random key sent by the originator.

5.3.3 Avoiding redundant announcements

The above protocol would work – although for the protocol to converge to the optimal solution, a router has to announce every route change. Every announcement triggers a potentially expensive computation, so we seek to omit unnecessary announcements. In particular, we want to avoid re-announcing a route if it has not ostensibly changed.

That said, how does a router know if a route it is using has really changed? Presuming the router has received new announcements itself, and undertaken a new SPC, it could make one of two decisions:

1. change the next hop – in which case it is perfectly obvious that a route change has occurred and that it should re-advertise; or

2. keep the next hop the same.

In the latter case, we must remember that the router has only local information, so it is not clear whether:

1. the new route is the same; or

2. the next hop is the same, but the route is different at some downstream point.

In the second case, we must re-advertise, because distances may have changed, and this may affect the decisions of other routers (even if the local next hop is the same). In the first case, we can omit re-announcing, and thus avoid overhead.

To enable a router to detect a route change in the second case, we have to extend the protocol – such that the destination router not only returns the

shortest path ID, but also the encrypted distance of this path. Now if the originator starts a new SPC for that destination, he adds the old distance to the request. The destination router can then compare the new distance with the old distance, and adds the result of the comparison to the response. The full response contains:

- The computation ID.
- The path ID.
- The path ID of the shortest path, encrypted with the encryption key sent by the originator.
- The distance of the shortest path, encrypted with the public key of the destination router.
- The "same distance" flag: a Boolean flag showing if the distance of the old and the new shortest route are identical, encrypted with the encryption key sent by the originator.

If the originator of the SPC receives a reply to its computation request, it decrypts the shortest path ID and the same distance flag. It will only update its routing table if it receives a reply for this computation through the path of the new shortest path, and then announce the update to its neighbours if either the new next hop is different to the old one, or if they are the same but the "same distance" flag is not set.

Note that, we only need to pass a "same distance" flag. We don't need a "same path" flag because the change is only important if it affects downstream decisions, and this will only be the case if the distance has changed.

5.3.4 Timeouts

For every SPC there has to be temporary routing information stored at routers. In order to reduce memory overhead, we need a mechanism to decide when it is safe to delete this information.

Also, since announcements for the same destination often arrive in close succession, we introduce a waiting period for the originator to start the SPC after receiving an announcement. This reduces the number of SPCs (at the cost of potentially delaying convergence).

In addition there is the time the destination waits for queries before computing shortest paths. In all there are three timers required:

1. `waitForAnnouncementsTime`: the time a router delays between receiving an announcement for a destination, and commencing the SPC.

2. `waitForRequestsTime`: the time that a destination waits from receiving a SPC request, before beginning a SPC response.

3. `waitForRepliesTime`: the time that information is kept in temporary storage for reverse-path lookups, in particular, the time the originating router waits for replies after creating an SPC request.

Figure 5.3.2 shows the timeline of these different timeouts. We discuss the choice of these parameters in Section 5.4.

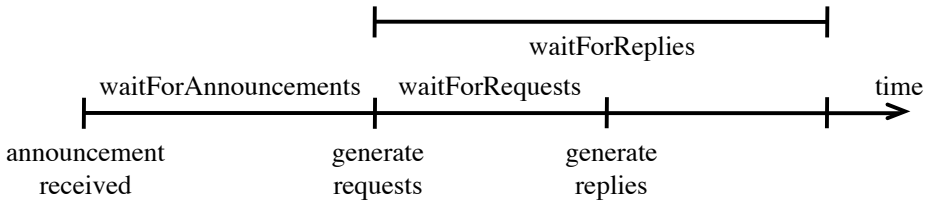


Figure 5.3.2: Timeline of `waitTime` timers.

Without timeouts STRIP behaves essentially like a distributed asynchronous Bellman-Ford algorithm, so the same argument for convergence applies (see [13, Chapter 5.2.4]). With the introduction of timeouts, convergence behaviour is not especially clear. For instance, if `waitForRequestsTime` or `waitForRepliesTime` are chosen too low, then no path with minimum transmission delay greater than those timeouts will ever be discovered. Transmission times for messages depend on processing, queuing and transmission

delays at in-between routers (which are all variable in nature), and we might envisage situations where this leads to long-term oscillation. However, most realised protocols have such timers, and standard practices (such as including jitter in timers) have generally been accepted as approaches to mitigate such problems, and accordingly, we use these here. In our experiments (detailed later) we saw no problems with long term oscillation, and we examine the correct choice of timers to avoid incorrect convergence problems.

5.3.5 Implicit Loop Detection

Since a SPC request only leads to a new routing table entry if the request travelled to the destination and back along the new shortest route, this entry must be loop free.

5.3.6 Privacy

The protocol has privacy-preserving properties in the *honest-but-curious* security model, i.e., if the parties correctly follow the protocol, there is no efficient, single adversary that can extract more information from the transcript of the protocol execution than is revealed by that party's private input and the results.

Topology information From any shortest path computation an originating router will only learn the next hop to the destination. But that is information it already knows, i.e., its neighbouring routers.

The destination router involved in the shortest path computation can also not learn any new information about the topology of the network, since the path information (the path, and the computation ID) is distributively stored in the memory of the routers of the path. Without collusion, this information is not obtainable. Also the origin ID is not revealed, as otherwise the destination router would know where the request came from, and

together with the different path lengths it could infer information about the networks topology.

Distance information No intermediate router in a shortest path computation can gain distance information, since this information is encrypted with the destination router's public key. Only the destination router is able to decrypt. Therefore the security is based on the security of the homomorphic encryption system. And although the destination router learns the distances of all paths, it cannot link this information to any router in the paths, since the only information about the paths it learns is simply the last hop. Note that all information about the originator of the request is anonymised.

5.3.7 Authentication

Our protocol provides destination authentication – that is, the originator of a request can be assured that the response was created by the destination router, and no one else.

For every shortest path computation the originator creates a random symmetric key K_s. It then encrypts K_s with the public key pk_D of the destination D and adds it to the request. Only D, the holder of the private key corresponding to pk_D, can learn K_s. But since K_s is necessary to create a valid reply, it could have only been the destination creating the reply. As such, the originator can be assured that the reply is fresh and not a replay, because every SPC has a new key K_s.

5.3.8 Possible attacks outside the security model

The privacy properties of STRIP only hold in the non-collusion honest-but-curious security model. It assumes that participants follow the protocol, though they may seek to learn additional information.

This is a reasonable assumption for a routing protocol – routing requires protocols to be followed correctly at some level, or it cannot reasonably be expected to work at all. That said, we do not want to conceal weaknesses of the protocol against more powerful attackers.

If we allow parties to deviate from the protocol, or to collude with other parties, they might mount the following attacks.

Sabotage Routers can sabotage the protocol in a number of ways. They can drop random packets (either control packets to sabotage computations, or data packets after the fact). They can use invalid input weights to manipulate the results. They can also perform a Denial of Service (DoS) attack. Since every announcement triggers a rather expensive shortest path computation, flooding the network with announcements potentially leads to overload.

The emphasis in our protocol is privacy, not protection from such attacks, which are in any case possible at present with most current protocols.

Attacks to gain information There are several ways in which a participant in the protocol might actively attempt to elicit additional information. Firstly, an originating router O can request multiple route computations with different subsets of its peers. The result is the next hop for each subset of peers, and from this O can deduce the order of the routes. A partial version of this may happen during route convergence, and so partial orderings are one form of leakage in this protocol, when performed multiple times. A more serious form of this attack involves O performing many such calculations and deliberately corrupting its component of the calculation by adding values to partial metrics it learns. This could potentially allow O to learn the actual distance metrics if performed enough times. However, for a metric with a reasonable range, this attack requires many computations and is unlikely to be accomplished unnoticed.

Secondly, multiple routers could collude to obtain more information. For instance, both the destination router, and other routers along a path sharing a common neighbour, can compute its distance value by decrypting the partial sum of the distances and taking the difference.

Finally, once routes are established, traffic will follow. An observer of traffic may be able to learn a great deal about the routes in a network (e.g., see [36, 124]). Furthermore, by changing its own weights, and observing traffic flows, a node may be able to learn a substantial amount. These type of attacks are unavoidable as long as the traffic paths are not hidden from the nodes, but there do exist anonymous forwarding schemes (e.g. [47]) that allow one to disguise sources/destinations and routes through a network. If such a scheme were designed for use on top of privacy-preserving routing, then we may be able to avoid this last type of information gathering attack, though typically at the expense of some loss of efficiency.

5.4 Evaluation

Cryptographic protocols usually create overheads, and in the case of STRIP there are messages passed above and beyond those of a typical PVP. Our first goal, therefore, is to determine the overheads of STRIP.

We can see two types of overhead: the additional messages passed (and the additional length of these messages in comparison to those of a PVP protocol), and the cost in terms of extra time to converge. There is an additional computational cost to the protocol, but we will account for that through the calculation of additional delays.

We assess the protocol through two means: simulation and implementation. The simulation is necessary because we do not have the resources to assess the performance of the real implementation in distributed hardware; as such, the simulation is used to show scaling of the protocol, and used for setting features (such as timers) that require many experiments. However,

the final proof of the pudding lies in the implementation.

We first discuss the simulation results. We wrote a discrete-event simulation of STRIP focusing on the processing capabilities of the routers in the network. We implemented the simulation using Python, and in particular the SimPy [1] package, a framework for implementing a process-based discrete-event simulation.

The routers are modelled to have a processing queue, unlimited in size, where the SPC-packets are processed sequentially, in the order of their arrival (FIFO). Each router can have one, or several, of these processing queues.

The STRIP protocol routers were configured with the following parameters:

- `requestTime`: the time a router needs to process a SPC request. This is dominated by the encryption. We set this value on all simulations to 4ms, the time for our Paillier encryption on a fairly standard CPU.

- `replyTime`: the time a router needs to process a SPC reply. That's just a lookup in the temporary routing table. We set replyTime to 0.1 ms.

We implemented the `waitForAnnouncementsTime` (wfat) as a random variable X with a uniform distribution and $1/2$ wfat $\leq X \leq 3/2$ wfat. If the waiting time is fixed it can lead to bursts of almost simultaneous shortest path computations which create load bursts on the routers. With the randomisation we achieve a more equal load on the routers, and avoid potential synchronisation effects.

We also implemented a simulation of the path vector protocol described in 5.3. For a fair comparison, the protocol includes an `announcementDelay` parameter similar to the `waitForAnnouncementsTime` in STRIP (routing protocols often have such a parameter, for instance the `minRouteAdvertTimer` in BGP). When a router learns a new route, it waits for `announcementDelay` before announcing the changes to its neighbours. The purpose is to prevent

load spurts, and it is randomised in the same way as the equivalent parameter of the STRIP protocol.

We test the protocols on several different networks whose topologies are described below. Every link between routers has a transmission delay of $5 + x$ ms, with x being sampled from an exponential distribution with mean 1. In each case, the routers are started at the same time, and start by announcing their immediate neighbours. This is the most stressful test of the performance of the algorithm.

For every configuration, we ran the simulation 50 times with different seeds for the random number generator – the reported figures are the averages over these simulations.

5.4.1 Comparison

The added privacy measures of STRIP introduce overhead compared with the PVP. Figure 5.4.3 shows the convergence time of STRIP vs. the path vector protocol in an Erdös-Rényi graph with $p = 10\%$ and respective number of nodes, and a Barabási-Albert graph with 2 new edges for each node.

We observe that the convergence times increase due to the additional message passing delays, but that the increase is approximately 20% on average over all the cases. If the computational times of the encryptions were reduced, then these overheads could be substantially reduced.

We are not suggesting that either type of network is realistic, but they do test opposite extremes. The former is a relatively regular network, while the latter has power-law degree, and this creates a small number of high-degree hub nodes. We can see in the results that both protocols converge more quickly on the Barabási-Albert graphs, due to the smaller search space (routes concentrate on the few hub nodes).

There is a more significant difference in the communication costs of the two protocols. Figure 5.4.4 shows the number of messages sent until convergence for the same graphs as in Figure 5.4.3. The number of announcements

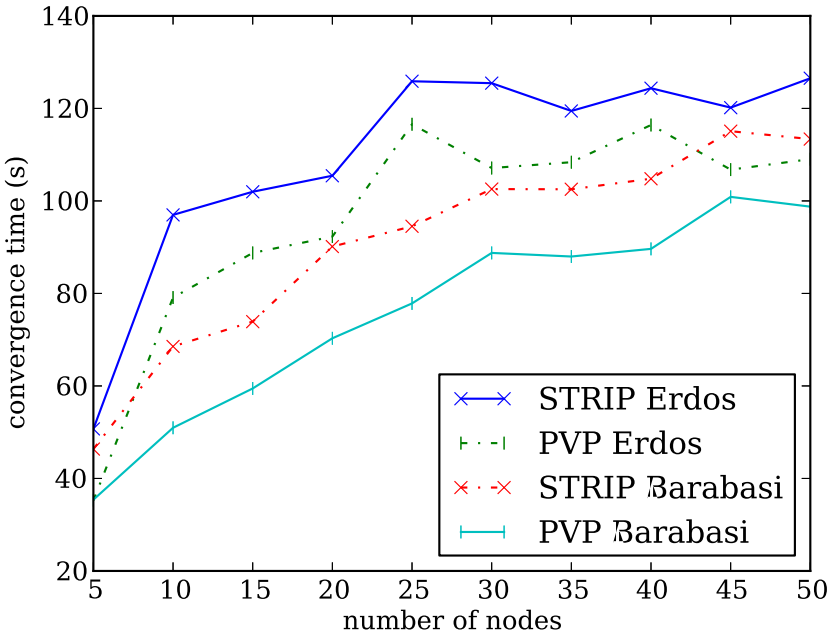


Figure 5.4.3: Comparison of the convergence time between STRIP and PVP for Erdös-Rényi and Barabási-Albert graphs.

made by both protocols are similar, but the SPC in STRIP introduces additional messages. The sizes of these messages are dominated by the cypher-texts; a request contains two, and a reply contains one cypher-text. With Paillier's encryption scheme (with a key size of 1024 bits), the cypher-text are 2048 bits in length. Thus requests are around 512 bytes, and replies are around 256 bytes. The extra, larger messages introduce a communications overhead, but it is still manageable simply because todays networks have an exponentially larger available bandwidth than those for which a typical PVP was designed (20 years or more ago).

However, the number of requests and replies grow significantly faster than the number of announcements for bigger graphs, since there are more paths to be evaluated and the average path length grows.

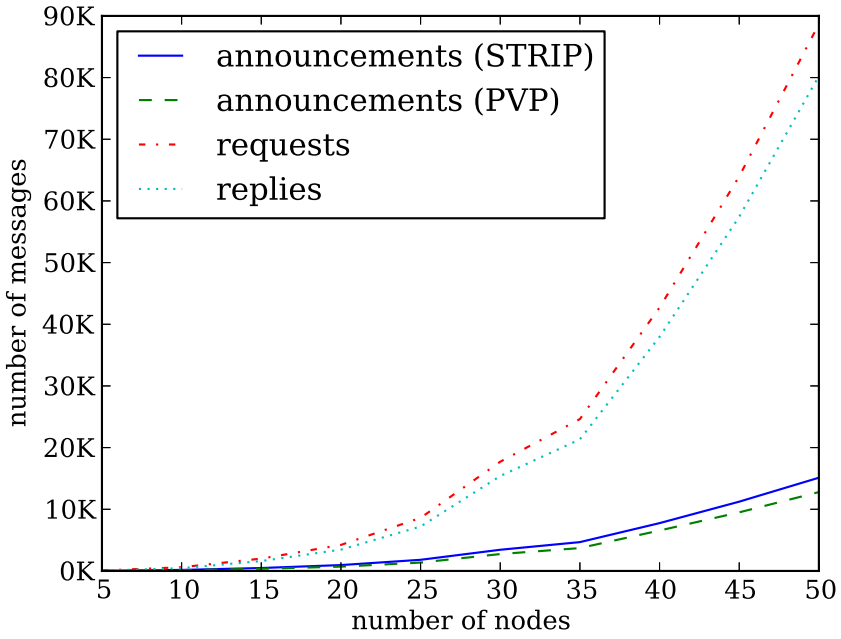


Figure 5.4.4: The number of messages sent until convergence in the Erdös-Rényi graph from Figure 5.4.3.

5.4.2 Parameter Choice

STRIP has three parameters that need to be configured:

`waitForAnnouncementsTime`,

`waitForRequestsTime`, and

`waitForRepliesTime`.

Their chronological sequence is shown in Figure 5.3.2.

The value for `waitForRequestsTime` depends on how long a request travels from the originator to the destination through the network. Too small, and requests are dropped; but large values slow down convergence. Figure 5.4.5 shows convergence time and deviation from the optimal routing solution for different values for `waitForRequestsTime` in an Erdös-Rényi graph with 30 nodes and $p = 15\%$.

Figure 5.4.5 shows that the convergence times (left axis) generally increase with an increase in the `waitForRequestsTime` parameter. This effect

can be dampened by an implementation trick we call *shortcut*. The originator knows how many requests it has created and therefore, after receiving a reply for every request it sent, it does not have to wait any longer because there cannot be any more replies. Sending the number of requests in each request to the destination router means it can stop waiting after it receives all the requests. Figure 5.4.5 shows convergence times with and without *shortcut*.

Figure 5.4.5 also shows the "deviation" (right axis) from the correct routing solution, which can be non-zero if too many routing messages are dropped. The figure shows that there is a minimum value for this parameter, above which the routes converge correctly. In all our simulations we found that choosing `waitForRequestsTime` to be $n(c+td)$, where n is the number of nodes in the graph, c the time it takes to compute an encryption and td the average transmission delay on a link, results in an optimal solution being found. This was a reasonable compromise between additional convergence time, and finding the optimal solution.

The value for `waitForRepliesTime` depends on the sum of the value for `waitForRequestsTime` and the times it takes for the replies to travel back to the originator. We choose `waitForRepliesTime` to be twice the `waitForRequestsTime`.

The `waitForAnnouncementsTime` parameter also has a dramatic influence on the convergence time. Choosing this to be too small leads to STRIP not finding the optimal solution. If it is too small, the routers receive too many requests to process – so queues may build up and at some point the number of requests may exceed the processing capabilities of the router leading to dropped requests. In contrast, larger values slow convergence. Figure 5.4.6 shows the convergence time of STRIP in an Erdös-Rényi graph with 30 nodes and $p = 15\%$. The effect of the deviation from the optimal solution can be dampened by allowing routers to process requests in parallel (in this measurement series we assigned a router one processing unit for every 4

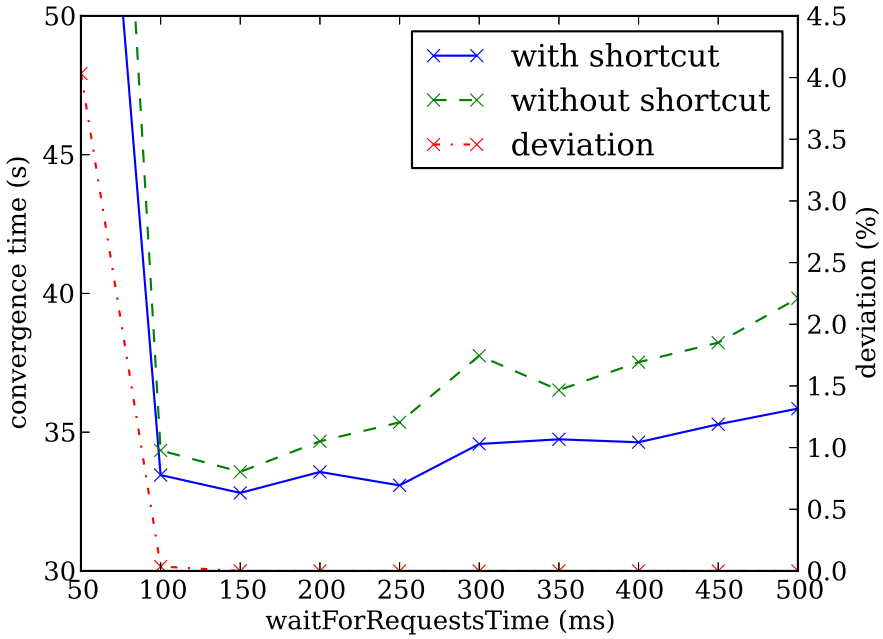


Figure 5.4.5: Convergence time and deviation from optimal routing solution for different values for `waitForRequestsTime`.

ports). The resulting convergence times are almost linear in the delay time.

5.4.3 Performance

The performance of the protocol, with regards to the smallest time to converge to the optimal routing solution, depends on the number of messages; this is because processing the cryptography in the messages represents the bottleneck. If the processing queue of a router fills faster than it can process the messages, it increases the overall round-trip time for a shortest path computation; in extreme cases, messages may even be dropped.

Obviously, the more routers in the network, the more messages that have to be processed, but the number of links in a network also has an influence on the number of messages sent. Figure 5.4.7 shows the number of messages in an Erdös-Rényi graph with 30 nodes with varying average node degree. The increase of edges in the graph increases the number of possible paths

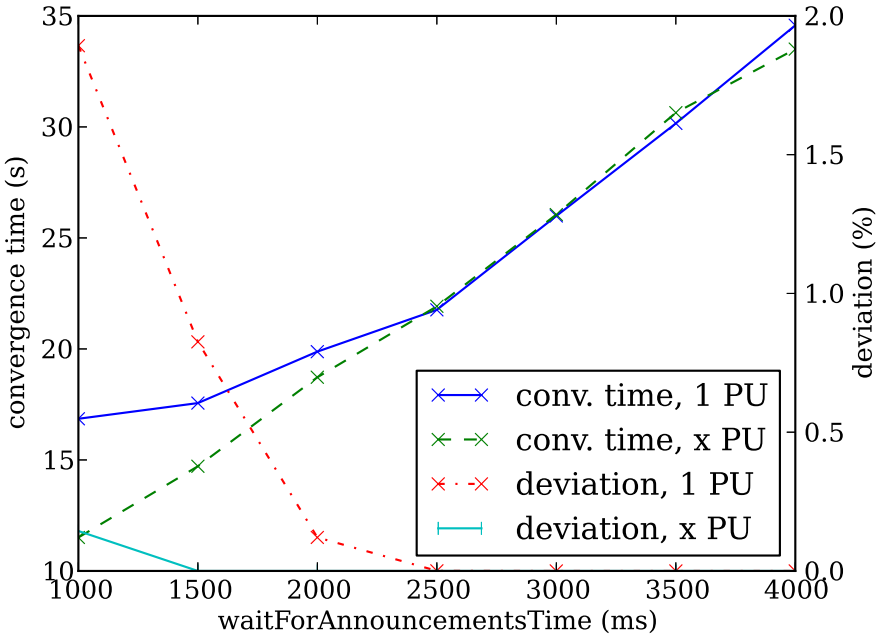


Figure 5.4.6: Comparing convergence time and deviation from optimal routing solution for different values for waitForAnnouncementsTime for single and multi processing unit (PU) routers. In the multi processing case a router gets assigned one processing unit for every 4 ports.

between nodes, which consequently requires that more paths have to be explored in each computation.

The overall amount of load for the routers is not the only determining factor for the protocol performance. It also depends on how this load is distributed over all routers. Figure 5.4.8 show the deviation from the optimal routing solution for an Erdös-Rényi graph with 30 nodes with an average node degree of 3 but different maximal node degrees. It shows that the nodes with high node degree are the bottleneck for performance of the protocol; this is because a higher node degree means being part of more paths, and therefore having to participate in more shortest-path computations.

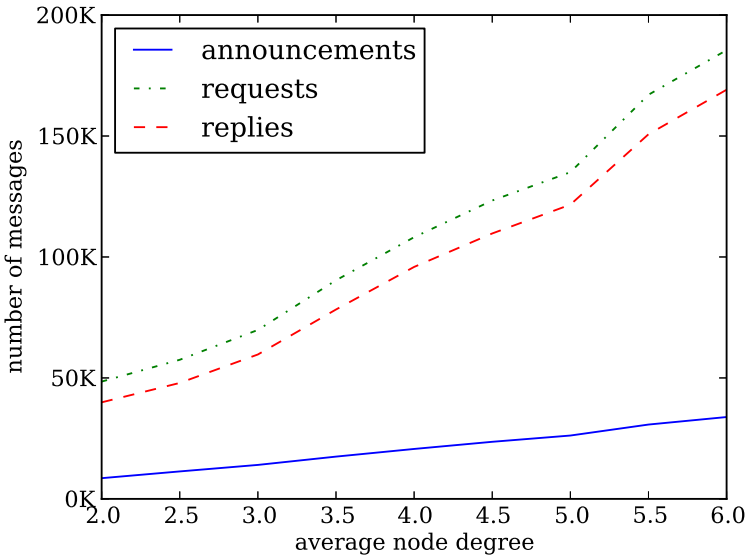


Figure 5.4.7: The number of messages for graphs with $n = 30$ but different average node degree.

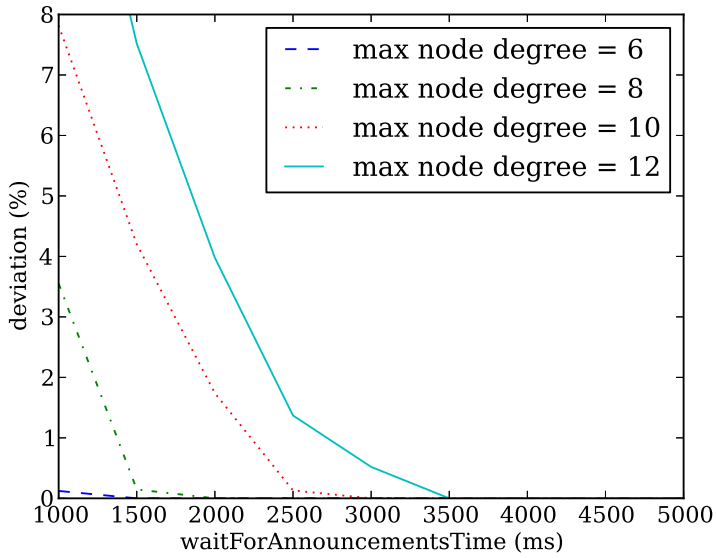


Figure 5.4.8: Deviation from optimal routing solution for graphs with $n = 30$ and same average node degree but different maximum node degree.

5.5 Implementation

We implemented the protocol using the Python programming language. We chose Python because of its suitability for rapid prototyping and its vast collection of libraries. The core of STRIP is *twisted* [2], an event-driven networking engine. It abstracts the underlying complexity of networking by providing a suitable set of primitives. It enabled us to implement STRIP in just 680 lines of code.

For example, setting up a TCP server listening for connections from other routers is achieved using just one line of code:

```
stripservice = internet.TCPServer(
          config.getint('server','port'),
          STRIPServerFactory(router))
```

Figure 5.5.9: Initialisation of a TCP server to listen for connections from other routers

The STRIPServerFactory, thus created, in turn creates a STRIPServer object for each connection. Processing messages simply requires overwriting of the stringReceived method.

```
class STRIPServer(NetstringReceiver):
          def stringReceived(self, data):
               ...
```

Figure 5.5.10: Definition of the STRIPServer class. Overwriting *stringReceived* enables message processing.

For the homomorphic encryption we chose Paillier's encryption scheme [114]. We based our implementation on the work of [51], but we used a wrapper module to be able to use the very efficient GNU multiple precision arithmetic library to speed up computations dramatically. We also improved

the performance of decryption by applying the Chinese Remainder Theorem (CRT). The computationally expensive part of the decryption is a modular exponentiation with a large exponent. The CRT enables us to divide the exponentiation into two exponentiations with much smaller exponents. The new exponents have just half the bit size of the original one. This method leads to an improvement of approximately a factor of 4.

The source code is available for download at
`https://github.com/wilko77/STRIP`.

5.5.1 Emulation

We emulated routers using AutoNetkit [86], which creates a *netkit* [120] lab. Netkit is an environment for performing network experiments with several virtual network devices that can be interconnected to form a network on a single PC. Given the network topology description in graphML format, it automatically generates the netkit configuration files to set up the virtual routers and the connections. We modified AutoNetkit to also generate the STRIP configuration files.

It is important to realise, though, that this is a real protocol stack, running on real router software (on virtualised router hardware). As such, our emulation experiments can demonstrate success of the protocol, and overhead in terms of messages – but convergence times for such a network are inaccurate, hence the need for the earlier simulations.

We used the following networks for the emulation runs. All networks consist of 11 nodes.

- Random tree. It has the smallest possible number of edges for a connected graph. We choose the weights of the edges uniformly at random between 1 and 10.

- Clique. Fully connected, thus maximum number of edges for a graph. Again, randomly chosen weights on the edges.

- The Abilene network. Weights on the edges are the distance of the edge in km.

Netkit starts the virtual routers in a sequential order, and only after the first one is fully running does it start the next one. We run an emulation until the routing converges, and then test the result for correctness. Table 5.5.1 shows the number of sent messages for the different networks.

network	#encryptions	#announcements	#other msgs
tree	512	91	584
fully connected	14,477	2,069	26,570
Abilene	1,286	192	1,954

Table 5.5.1: Comparison of the number of messages until convergence.

As seen with the simulations, the density of a network is a determining factor for the communication and computation costs. The difference between the best case (tree) and worst case (clique) for the same size network is significant. However, real world networks are rather sparse, as every link introduces more costs and does not necessarily result in better performance.

5.6 Conclusion

This chapter presented STRIP, a shortest-path routing protocol, which does not reveal the length of paths to its participants. It opens up a set of operations that could enhance privacy (and hence security) in future protocols.

The protocol has limitations: it introduces overheads, and doesn't implement all of the features that one might like to see in a modern, BGP-like protocol. However, the basic components of the protocol are easily extensible; indeed, the homomorphic encryption can implement other types of path metrics, or combinations of them.

In the next chapter we apply SMC techniques to fraud detection in telecommunication networks.

Chapter 6

Privacy-Preserving Fraud Detection Across Multiple Phone Record Databases

After looking at a network management problem in computer networks in the previous chapter, we now focus on a somewhat different problem in telecommunication networks.

Subscription fraud, *i.e.*, customers signing up to a service with no intent to pay, causes significant losses in the telecommunication industry. Telecom operators have developed strategies to identify those fraudsters, but fraudsters tend to migrate from one carrier to another. Data sharing between telecom operators would increase fraud detection rates, but phone records are protected by law and operators might be reluctant to share information about fraudsters because they see it as giving a competitive advantage.

We propose several protocols based on different SMC techniques to enable fraud detection across multiple databases without revealing additional information. We also propose a model to generate phone records, with which we evaluate how the choice of parameters affects detection performance. We show feasibility and a comparison of performance and costs with implementations of our protocols.

6.1 Introduction

Fraud – the deliberate practice of deception in order to secure unfair or unlawful gain – is as old as humanity itself and is quick to adapt to new technologies.

Experts estimate the losses in the telecommunication industry due to fraud for 2013 at 46.3 billion US dollar, approximately 2% of the global telecom revenue [32].

Fraud in the telecommunication industry comes in many different forms with subscription fraud being the biggest concern for telecom operators [121].

The characteristics of subscription fraud are that someone signs up for a new service (e.g., a new phone, extra lines, ...) with no intent to pay. All calls over this line are therefore fraudulent but consistent with the profile of the user, though fraudsters often hide behind false identities to hamper fraud detection efforts.

To combat fraud, and minimise losses, telecommunication companies use different fraud detection techniques, with the aim to identify fraud as quickly as possible.

However, the phone record databases are huge and complex, and they contain only a relatively small number of fraudulent entries, which makes the extraction of the relevant information for fraud detection challenging.

Several techniques for fraud detection have been proposed in recent years. Bolton and Hand [21] took a statistical approach, Phua *et al.* [117] showed an overview of data-mining based techniques, and Becker *et al.* [9] proposed a graph-based solution. We extend their approach as it has proven to be useful in supporting the fraud investigators at AT&T.

All three fraud detection techniques have in common that they only consider data from one source. But the telecommunication market is heterogeneous, *e.g.*, in the U.S market no company had a share greater than 35% of carrier subscriptions in 2013 [37]. Fraudsters might migrate from one

carrier to another, once their initial fraud is detected. How can the other carrier identify this fraudster with no prior data? After all, they look and behave just like new customers.

Obviously, the chances of detecting fraud would increase if the telecommunication companies shared their information about fraudsters. However, phone data is protected by law and information about fraudsters might be seen as providing a competitive advantage, so at present this data is not shared anywhere, to the best of our knowledge.

We present a solution to detect subscription fraud that allows sharing fraudster information without revealing any information from the databases other than fraud matches.

Our contributions are:

- Real phone records are protected by law. So we propose a model to generate synthetic phone records with nontrivial relationships.

- We investigate how the choice of matching criteria and classifier affects the detection performance of the graph-based fraud detection approach by Becker *et al.* [9].

- We extend the graph-based fraud detection approach of [9] with *two* different secure multiparty computation techniques to construct privacy-preserving fraud detection protocols for multiple parties.

- We implement the protocols and compare their communication and computation costs.

6.2 Background

There are many different forms of fraud in the telecommunication industry. Examples of common varieties of fraud, as described in [9], are as follows:

- *Subscription fraud.* Someone signs up for a new service (e.g., a new phone or extra lines) with no intent to pay. All calls over this line are fraudulent but consistent with the profile of the user.

- *Intrusion fraud.* An existing, otherwise legitimate account is compromised in some way by an intruder, who subsequently makes calls on this account. In contrast to subscription fraud, the legitimate calls are interspersed with fraudulent ones.

- *Fraud based on loopholes in technology.* Technology used in conjunction with the service can have vulnerabilities. Consider voice mail systems as an example. Some can be configured to allow calls to be made out of the system. If fraudsters are able to exploit a vulnerability they may then use it to make outgoing calls.

- *Social engineering.* Psychological manipulation of people into performing actions or disclosing information to gain access to a customer's account. In 2006 this method was used to gain access to the private phone records of the board members of HP [15].

- *Fraud based on new technology.* New technology, such as the Voice over Internet Protocol (VoIP), enables telephony at very low cost. Fraudsters realised that they could purchase the service at a low price and then resell it illegally at a higher price to consumers.

- *Fraud based on new regulation.* New regulations can have unintended loopholes which allow fraudsters to exploit these regulations to their advantage. See for example the payphone compensation rules from 1996 [53].

- *Masquerading as another user.* For instance, stolen credit card numbers can be used to place calls masquerading as the card holder.

Privacy-preserving computation has been applied to fraud detection techniques before. Vaidya *et al.* [137] proposed a privacy-preserving outlier detection protocol. The aim is to detect unusual behaviour (*e.g.*, if an intruder compromises an account and starts to make calls on this account the calling patterns changes unexpectedly.) Our fraud scenario is different, as we want to be able to detect the same calling pattern in different accounts. Grosskreuz *et al.* [69] proposed a secure protocol for top-k subgroup discovery on horizontally partitioned data. It finds patterns in the union of databases, and the quality of every subgroup depends on all databases. In our setting the databases are independent and the quality of the match only depends on the corresponding database.

In this work we focus on subscription fraud, as it alone amounted to a loss of 5.22 billion US dollars in the telecommunication industry in 2013 [32]. The challenge in detecting subscription fraud is that without prior data, it is hard to distinguish between legitimate customers and fraudsters, since the only difference is that fraudsters have no intent to pay for the service. Furthermore, because fraudsters tend to hide behind false identities, a database with identities that showed fraudulent behaviour in the past is not likely to be very effective for fraud detection.

However, although fraudsters might change their identity, their calling pattern (the numbers they call, frequency and length of calls) often stays the same, since this pattern is defined by their social network.

Hence, a more promising strategy to identify fraudsters is to look at their usage signature rather than looking at their claimed identity. Becker *et al.* [9] did exactly this by computing so called *Communities of Interest* (COI) signatures of the phone usage of subscribers. They contain the top-k communication partners of a subscriber. The idea behind this is that signatures are different for different individuals but fairly stable over time. This can be explained by people having individual social networks and large changes in these are rare. Knowing COI signatures of fraudsters helps telecommuni-

cation companies to identify fraudulent new subscribers by matching their COI signature to those in the fraudster database, as shown for real phone data at AT&T in [9].

6.2.1 Communities of Interest

If we look at phone records as a graph, in which nodes are phone numbers and directed edges represent communication between those numbers, then we can see that, although the total number of nodes in the graphs might be in the millions with billions of edges, the number of direct neighbours of an individual node is usually only a few. To create a signature of an individual's calling behaviour we use a framework called the *Community of Interest (COI)* signature [38]. This signature consists of the top-k numbers called by the target number and the top-k numbers that call the target number.

Although ideally the COI signatures would be stable we also have to allow for changes in the call-graph network such as changes in phone numbers, or in the social network of an individual. The COI framework accounts for this by computing a moving average over the call-graph network over time.

The call-graph network can be represented as a weighted, directed graph G, with nodes representing the phone numbers and edges that represent calls between two nodes (denoted by $N(G)$ and $E(G)$). Each edge $e \in E(G)$ is assigned a weight $w_G(e)$ that describes the aggregation of the transactions between the two nodes. This can be the number of calls between the nodes, the sum of the durations of the calls, or other call-based properties. We use number of calls as edge weights, as our simulation does not generate other call properties.

To describe the historical behaviour of the graph, we first have to define a graph operator $\oplus$ that computes the weighted sum of two graphs A and B, such that if $G = \alpha A \oplus \beta B$ then the nodes and edges of the new graph G

are the union of the nodes and edges of the graphs A and B,

$$N(G) = N(A) \cup N(B),$$
$$E(G) = E(A) \cup E(B),$$

and the weight of the edges of the new graph are computed as

$$w_G(e) = \alpha w_A(e) + \beta w_B(e), \text{ for all } e \in E(G),$$

where

$$w_A(e) = 0, \text{ for all } e \notin E(A), \text{ and}$$

$$w_B(e) = 0, \text{ for all } e \notin E(B).$$

We use a discrete time representation, that is, the time is divided into equidistant time intervals. Let g_t be the graph corresponding to the transactions during the finite time interval t, and G_t be the graph representing all transactions up to and including t, then $\hat{G}_t$ is called the top-k approximation of G_t which is defined by

$$\hat{G}_t = \text{top-}k\{\theta g_t \oplus (1-\theta)\hat{G}_{t-1}\},$$

where "top-k" is a pruning function that includes the k edges of each node with the highest weights. Everything not included in the top-k edges gets aggregated into an overflow bin called *other*.

The pruning function ensures that only the most relevant nodes will appear in the COI signature. But since calling behaviour is heavily skewed such that most of an individual's calls are made to only a few numbers, we can choose the parameter θ and k of the COI framework such that typically 95% of all communication behaviour is accounted for in the top-k edges. For a thorough discussion of the choice of parameters, see [72]. Pruning also reduces the size of the data we need to track to k entries per subscriber.

These COI signatures are used to detect subscription fraud in the following way. The signatures of known fraudsters are stored in a database, and

after some time T a new customer's signature is compared to the database. The idea behind this approach is that if a fraudster signs up to a new line they will most likely have a very similar signature to their past one.

The comparison of two signatures is a two-stage process. We have to define a measure of closeness, which we call *matching criteria*. We discuss different choices in Section 6.4.1. Furthermore we need a classifier to decide if two signatures are "close enough" to be considered to have originated from the same individual. Section 6.4.2 discusses different classifiers.

We consider different matching criteria/classifier pairs because different pairs suit different types of secure multiparty computation.

6.2.2 Secure Multiparty Computation

Again, we use secure multiparty techniques to secure the privacy of the parties' inputs. The protocols we propose in this chapter are either based on two-party secure function evaluation, which we describe in Section 2.3.6, or based on the private set intersection protocol described in the next section which uses homomorphic encryption. We describe homomorphic encryption in Section 2.2.

Private Set Intersection (Freedman *et al.* [57])

This is a two-party protocol between a client C and a server S. Each holds a set of inputs drawn from the same domain. At the conclusion of the protocol, C learns which elements of the sets are shared by both C and S. But neither party learns any element of the opposite party's set that is not an element of the intersection. Freedman *et al.* [57] present different versions of the protocol which are secure against *semi-honest* or *malicious* adversaries. We implement the semi-honest case and leave the malicious case to the interested reader.

A *semi-honest* adversary is a participant in the protocol that correctly follows the protocol but tries to gain more information about the other

party's input than can be inferred by its own private input and the output of the protocol. A protocol is secure in the semi-honest model if there is no *efficient* semi-honest adversary, *i.e.*, an adversary that runs in polynomial time.

A detailed description of the protocol is shown in Protocol 1. The underlying idea of this Private Set Intersection protocol is that C defines a polynomial P whose roots are the inputs $(x_1, \ldots, x_k)$, *i.e.*,

$$P(y) = (x_1 - y)(x_2 - y)\ldots(x_k - y) = \sum_{u=0}^{k} \alpha_u y^u.$$

C sends the homomorphically encrypted coefficients of the polynomial to S. Then S can use the homomorphic properties of the encryption system (in this case the addition of two encrypted values and the multiplication of a constant with an encrypted value) to evaluate the polynomial for each of its inputs. The key observation is that for S's input y the polynomial will evaluate to $P(y) = 0$ if and only if y is in the set of inputs of C. The server S will use this fact to hide its inputs. For every element y of its input set it computes $\mathrm{Enc}(r \cdot P(y) + y)$, with r being a fresh random number. Thus, for each element of the intersection of the two parties input sets the result of this computation is the corresponding element, whereas for all other values the result is random.

6.3 Test Data

Real phone records are protected by law in most countries. Therefore we have to simulate phone records to test our ideas.

Our aim is to find a model to create phone records that has similar characteristics to real-world data from [72, p. 15]. We aim to make the model as simple as possible though, in order to aid interpretation of results. In particular our model should only have parameters that have real-world equivalents, and for which values can be obtained by publicly available data.

INPUT: C's input: $X = \{x_1, \ldots, x_k\}$, S's input: $Y = \{y_1, \ldots, y_k\}$

1. (a) C uses interpolation to compute the coefficients of the polynomial $P(y) = \sum_{u=0}^{k} \alpha_u y^u$ of degree k with roots $\{x_1, \ldots, x_k\}$.

   (b) C encrypts each of the $k+1$ coefficients and sends the encryptions $\{\text{Enc}(\alpha_0), \ldots, \text{Enc}(\alpha_k)\}$ to S.

2. For every $y \in Y$:

   (a) S uses the homomorphic properties to evaluate the encrypted polynomial at y. That is, S computes $\text{Enc}(P(y)) = \text{Enc}(\sum_{u=0}^{k} \alpha_u y^u)$.

   (b) S chooses a random value r and computes $\text{Enc}(rP(y) + y)$.

   S randomly permutes this set of k ciphertexts and sends them to C.

3. C decrypts all ciphertexts and outputs all $x \in X$ for which there is a corresponding decrypted value.

Protocol 1: Private Set Intersection Protocol

The basis of our simulation is a graph representing the social network of a telco customer. Nodes represent the phone number of a customer and links between nodes represent the possibility for a phone call. Each edge is assigned a probability. To create a data point for a phone record, we iterate over all edges and generate a call between each pair of connected nodes with its probability, independently of all other calls. And since we want to allow more than one call between two nodes in a day, we do this iteration several times for a phone record for one day. We describe the details below.

6.3.1 Weighted Social Network Graph

Nanavati *et al.* showed in [107] that the node degree of call records fits a power law distribution. In order to achieve a power law distribution we modelled the social network of the subscribers as a Barabási-Albert graph. The graph is constructed by using a preferential attachment mechanism. Each new node is connected to m existing nodes with a probability proportional to the number of links of the existing nodes. Let S be the social network graph, we then assign each edge $e_{ij} \in E(S)$ between nodes i and j a weight v_{ij}, denoting the average number of calls per day between i and j. We chose the weights v_{ij} such that $E[v] = c/\mathrm{nd}(S)$, with $\mathrm{nd}(S)$ denoting the average node degree in graph S, in order to be able to create phone records with an expected average call rate per customer per day c. Given no better information about the distribution of the individual call rates, we chose the uniform distribution, according to Laplace's principle of indifference. Thus, we set the weights to $v_{ij} = \frac{2c}{\mathrm{nd}(S)} r_{ij}$ with r_{ij} being chosen uniformly at random between 0 and 1.

6.3.2 Phone Records

The phone records for a time period t can be described as a transactions graph g_t. We generate g_t from S such that $N(g_t) = N(S)$ and $E(g_t) = E(S)$.

To generate the edge weights w_{ij}, denoting the number of calls between i and j during time t, we first divide the time period t into d slots, and each slot can either have a call or not. The probability for a call in a slot is $p_{ij} = v_{ij}/d$, thus, $E[\text{no. of calls btw. } i \text{ and } j] = dp_{ij} = v_{ij}$. The weight w_{ij} is then set to the sum of all calls in the slots.

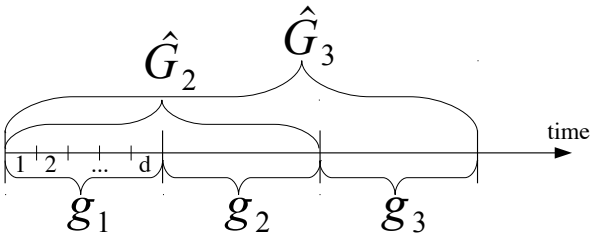


Figure 6.3.1: Timeline of the creation of the daily transaction graphs g_i and the historical transaction graph approximations $\hat{G}_i$

6.3.3 Validation

We generated phone records for 100 000 subscribers. We set the call rate c to the reported call rate of the US in 2007 [48, Section 24, Table 1146] of 5.2 calls per day per subscriber.

We compare our simulated records with real AT&T data. In [72, p. 15] Hill *et al.* show descriptive statistics of 1092 subscribers. The plots show cumulative distributions for total calls and edge degree for the subscribers over a 12 month period.

Figures 6.3.2, 6.3.3, 6.3.4 and 6.3.5 show these statistics for our data. Although similar there are some differences. The biggest difference is within the ultra-low usage customers. The data in [72, p. 15] starts at no phone calls per customer per year, whereas our model does not generate such low-usage customers. This might be explained by our data being generated under the assumption that every customer is active every day in the year, but in reality customers change providers and so may not be present for a part of the interval.

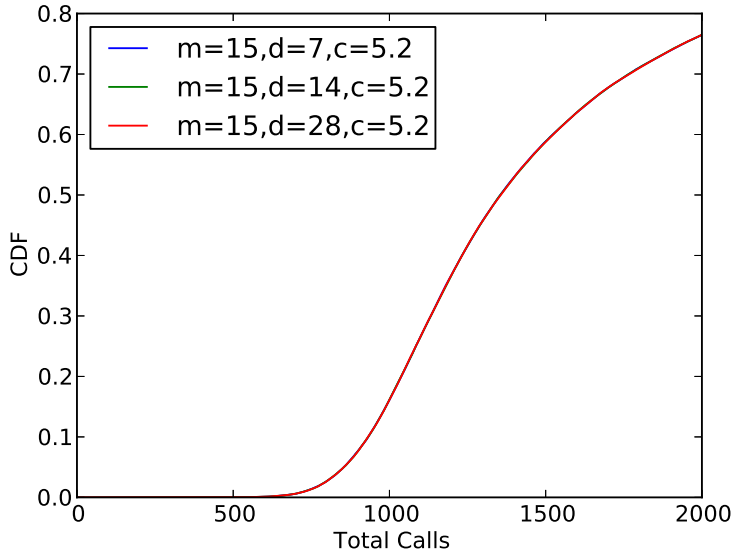


Figure 6.3.2: Cumulative distribution of total calls for call data generated with av. node degree $m = 15$, call rate $c = 5.2$ and various values for iterations per day d. All curves overlap, so choice of d has little effect.

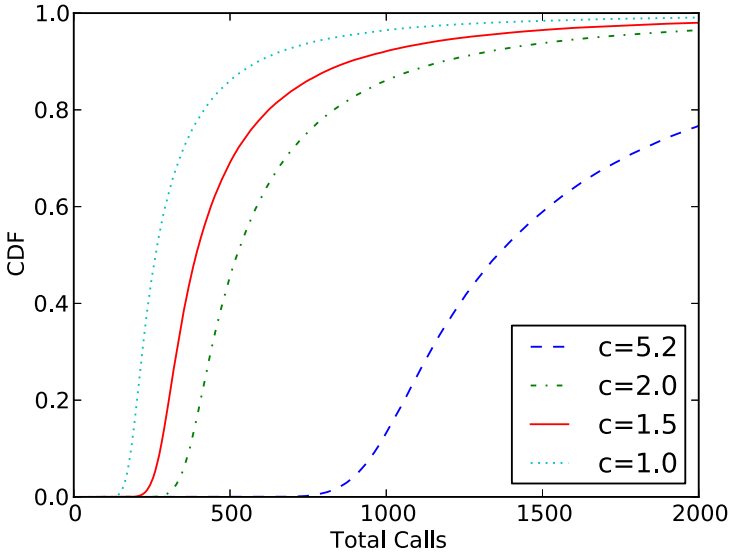


Figure 6.3.3: Cumulative distribution of total calls for call data generated with $m = 30, d = 7$ and various values for c.

Their graph suggests that the average daily call rate per customer is somewhere between 1 and 2. However, for our experiments we use the average daily call rate $c = 5.2$, the reported call rate in the US in 2007 [48, Sec. 24, Tab. 1146]. The difference might be explained by the rather small sam-

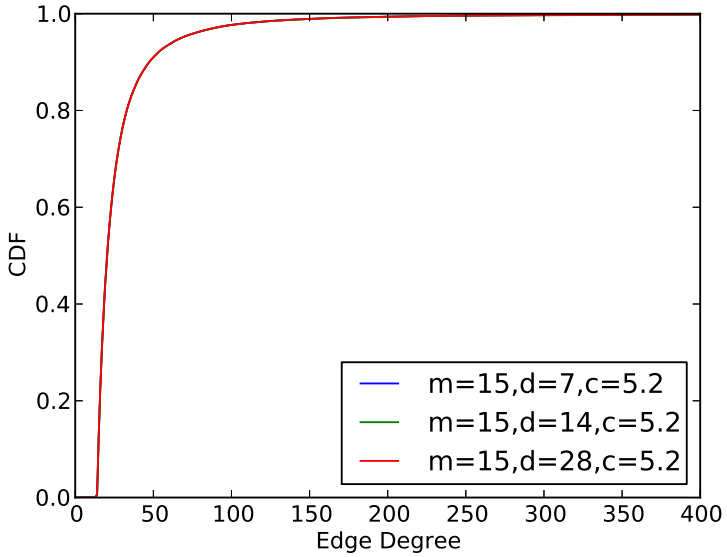


Figure 6.3.4: Cumulative distribution of edge degree for call data generated with $m = 15, c = 5.2$ and various values for d. All curves overlap, so choice of d has very little affect.

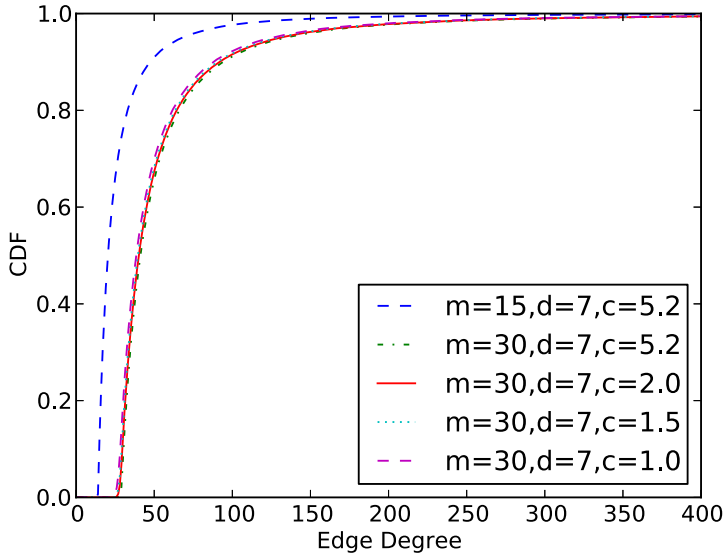


Figure 6.3.5: Cumulative distribution of edge degree for call data generated with $d = 7$ and various values for c and m

ple size of just 1092 subscribers out of the millions of AT&T customers. For the edge degree we chose $m = 30$ for our experiments, as this is the best match between Figure 6.3.5 and [72, p. 15]. The choice of d for the daily iterations does not seem to have a significant effect on the statistics as

Figures 6.3.2 and 6.3.4 show that the lines for different values for d overlap. We chose $d = 7$ for our experiments as a compromise between computation costs and detail.

6.3.4 COI Generation

We generated phone records for 100 days. For our experiments we need three different COI signatures for each subscriber. A signature of the first 90 days, representing the historical phone usage of the subscriber, a signature of day 91 and a signature of the ten days from 91 to 100. We choose the set of parameters for the COI framework as in [39]. The decay rate for historical data is set to $\theta = 0.9$, the number of entries in a COI is set to $k = 9$, and the pruning threshold to $\epsilon = 0.1$.

Although Hill *et al.* showed in [72] that tuning those parameter to your specific phone data might give better detection rates, we want to refrain from doing so, because in our application scenario we have different sets of phone records from the participating providers which most likely will have different sets of optimal parameters.

6.4 Matching

Matching two COI signatures, that is deciding if two signatures originated from the same subscriber, contains two components. A matching criteria that provides a measure of how similar these two signatures are, and a classifier that decides if we have a match.

6.4.1 Matching Criteria

Having two COIs we now need a measure of how likely it is that these two COIs originated from the same subscriber. We compare the following matching criteria:

- *Dice's Coefficient:* Proposed by Dice [45] to measure the amount of ecological association between species, his coefficient measures the similarity of two sets. Its values are bounded between 0 and 1, with 0 when there are no similarities and 1 if the two sets are identical. Let A and B be the sets of the phone numbers of COI_a and COI_b respectively. Dice's coefficient is defined as:

$$\mathrm{D}(\mathrm{COI}_a, \mathrm{COI}_b) = \frac{2|A \cap B|}{|A| + |B|}.$$

Dice's coefficient ignores weights, but a COI has weights assigned to it, so we want to investigate how using the weight information affects the detection performance.

- *Weighted Dice Coefficient:* Becker *et al.* [9] used a weighted version of Dice's coefficient for fraud detection. It uses normalised weights. Let $w_a(o)$ be the weight of phone number o in COI_a, and $p_a(o) = w_a(o) / \sum_{i \in A} w_a(i)$ be the normalised weight of phone number o.

$$\mathrm{WD}(\mathrm{COI}_a, \mathrm{COI}_b) = \frac{\sum_{o \in A \cap B} p_a(o) + p_b(o)}{1 + \sum_{o \in \text{top-}k} p_a(o)}$$

- *Hellinger Distance:* The second criterion used in [9] is based on the Hellinger Distance [12], which was designed to measure distances between statistical distributions. It also uses normalised weights.

$$\mathrm{HD}(\mathrm{COI}_a, \mathrm{COI}_b) = \sum_{o \in A \cap B} \sqrt{p_a(o) p_b(o)}$$

This sum is also bounded by 0 and 1.

- *Overlap:* Cortes *et al.* [38] proposed a measure called Overlap, with

$$\mathrm{Overlap}(\mathrm{COI}_a, \mathrm{COI}_b) = \sum_{o \in A \cap B} \frac{w_a(o) w_b(o)}{w_o},$$

where w_o is the overall weight of node o (the sum of the weights of all edges originating in o). The denominator corrects for the popularity

of an overlap node o, giving connections to less popular nodes more importance. The underlying idea is that high-use nodes like telemarketing shops or customer service numbers contribute less information to a unique fingerprint than low-use nodes. Again, a value of 0 stands for no similarities between the two signatures, but there is no general upper bound if the two are identical. Furthermore the interpretation of a value greater than 0 is not clear, since not only the intersection, but also the popularity of the nodes in the intersection have an influence.

6.4.2 Classifier

We want to test if a COI belongs to a fraudster. In our setting one telecommunication provider has a database of known fraudsters and another provider wants to test its new customers against this database.

The classifier has to decide, given the database of known fraudsters and the COI of a suspect, if this suspect is a fraudster or not.

We use two classifiers:

- *Threshold classifier:* If the value from the matching criterion of a COI from the database and the COI of the suspect is above a threshold s, the classifier declares the suspect as fraudster.

- *Delta classifier:* This classifier computes the matching scores for all elements in the database with the suspect COI and then looks at the difference between the highest and the second highest score. If this difference is above a threshold s, the classifier declares the suspect as fraudulent.

6.4.3 Comparison

We first generate phone records for 100 days and then compute the COIs for the first 90 days, the COIs for the 91^{st} day and the COIs for the 10 days from day 90 to 100. We randomly divide the 90 days' COIs in two groups:

fraudsters and non-fraudsters. The probability of being a fraudster is 0.05 and 0.95 for the non-fraudsters, respectively. We then sample 2000 COIs out of the 1 and the 10 day COIs for classification.

Matching Criteria

We compare the proposed metrics using a threshold classifier.

Figures 6.4.6 and 6.4.7 show the receiver operating characteristic for the different matching criteria for the matching of a 1-day COI signature and a 10-day COI signature to the database, averaged over 200 samples.

A Receiver Operating Characteristic, or simply ROC curve, is a plot which shows the performance of a classifier system as its threshold is varied. It plots the fraction of true positives out of the total actual positives (true positive probability) vs. the fraction of false positives out of the total actual negatives (false positive probability) at various threshold settings.

The detection performance with 10-day COIs is significantly better than the detection with 1-day COIs, which is expected, as a phone record has on average only 5.2 entries per day. However, the performance of the Overlap Criterion does not improve in the same way as the performances of the other criteria. We believe that this can be explained by the fact that the Overlap Criterion is not bounded between 0 and 1.

At least with our data, it seems that adding weight information to the matching criteria does not improve detection performance, but Becker *et al.* [9] showed that doing so achieved better results on real phone data.

Although the detection performance for 1-day signatures is not very high we have to keep in mind that in a realistic fraud detection scenario every classified fraudster has to be verified manually by an investigator. It is therefore essential that the false positive probability is very low to avoid unnecessary workload for the investigation team. Moreover, subscription fraud causes billions of dollars of losses each year, so even identifying a fraction of the fraudsters will lead to significant savings. Extending the time period for sig-

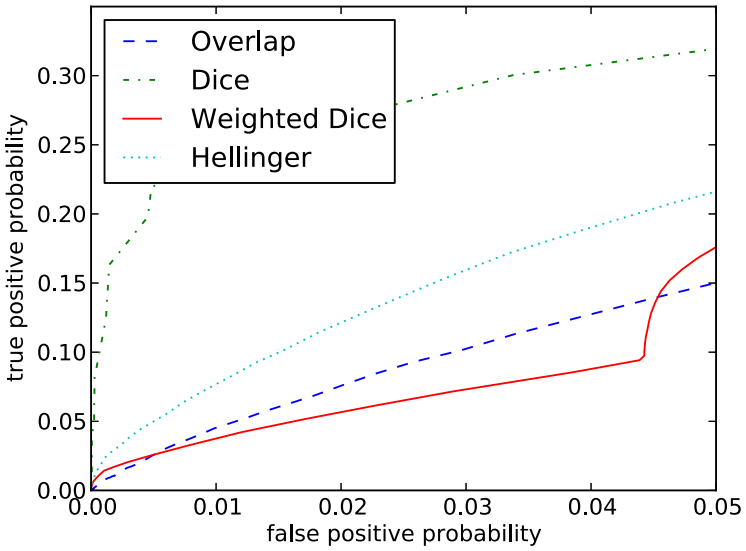


Figure 6.4.6: Comparison of the matching criteria for matching a 1-day COI to a database of 90-day COIs.

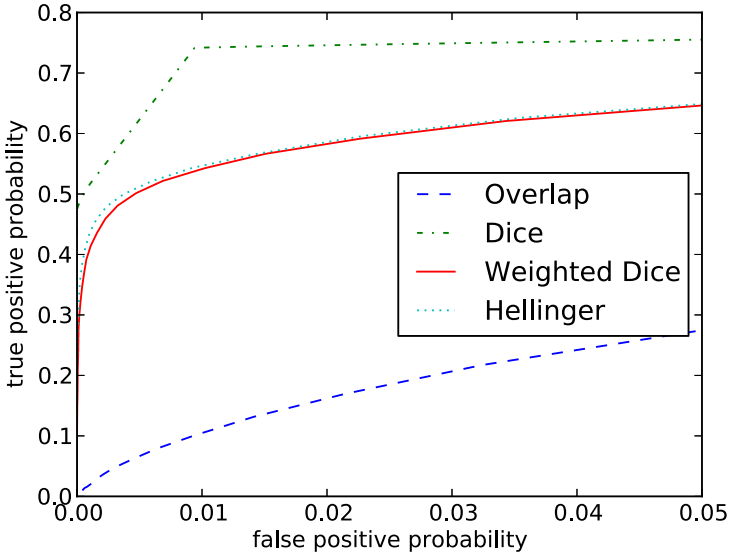
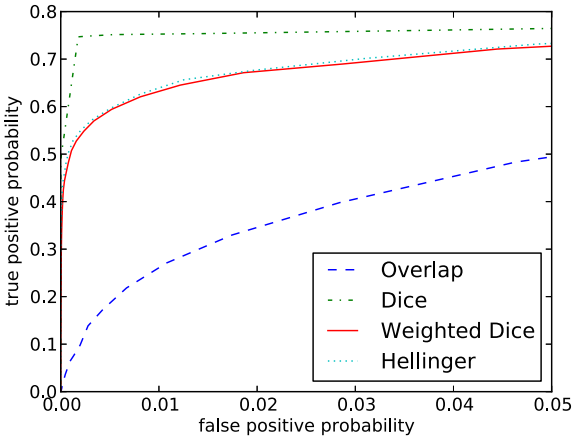


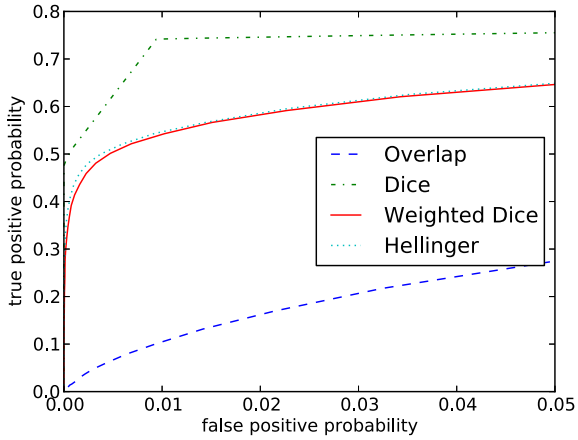
Figure 6.4.7: Comparison of the matching criteria for matching a 10-day COI to a database of 90-day COIs.

nature generation improves the detection probability significantly, after 10 days we are able to detect about half of the fraudsters with a false positives probability of less then 0.5%.

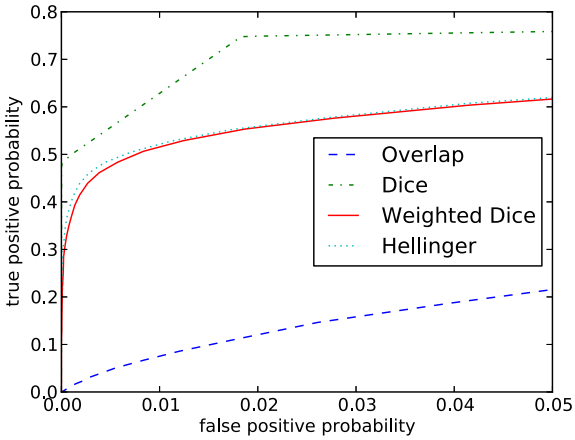
We arbitrarily chose the probability of a subscriber being a fraudster as 5 percent. However, changing this probability does not alter the overall



(a) 1 percent fraudsters



(b) 5 percent fraudsters



(c) 10 percent fraudsters

Figure 6.4.8: Comparison of the detection performance for different percentages of fraudsters amongst the subscribers.

trend. Figure 6.4.8 shows the comparison of the detection performance for 1, 5 and 10 percent fraudsters. In our experiments, the detection performance improves when fraudsters are uncommon (the most likely case).

Classifier

Figures 6.4.9 and 6.4.10 show the ROC curves of the comparison of the two classifier using the weighted Dice coefficient matching criteria. They show the fraction of true positives out of the total actual positives vs. the fraction of false positives out of the total actual negatives at various values for the classifier parameter s.

With all of our tested matching criteria the threshold classifier outperforms the delta classifier.

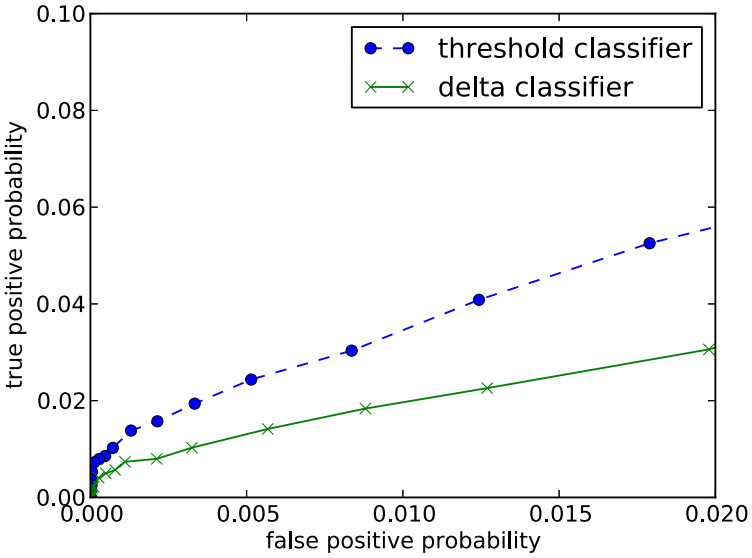


Figure 6.4.9: Comparison of the classifier for matching a 1-day COI to a database of 90-day COIs.

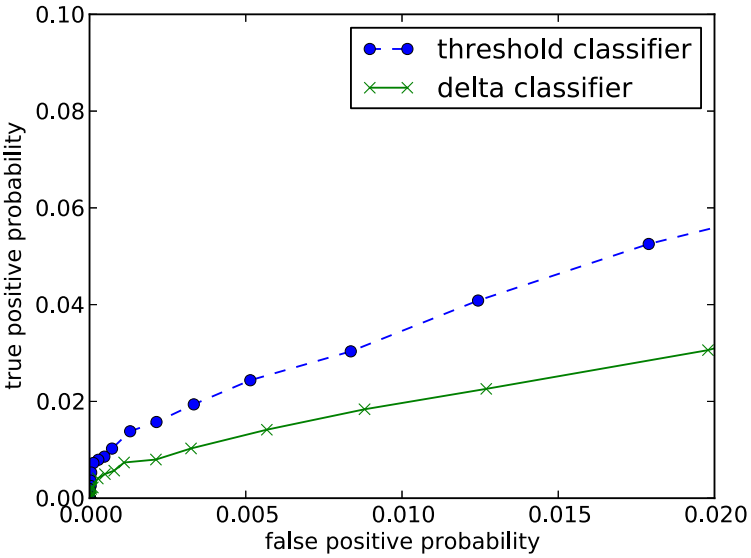


Figure 6.4.10: Comparison of the classifier for matching a 10-day COI to a database of 90-day COIs.

6.5 Privacy-Preserving Matching

In this section we describe two protocols with variations for enabling two parties to perform COI-based fraud detection with different privacy guarantees. One protocol is based on the Private Set Intersection protocol of Section 6.2.2 and the other is based on secure two-party computation of Section 2.3.6. The two methods have different levels of privacy and overhead.

First we present a protocol for doing matching with the unweighted dice criterion.

Although the inclusion of weight information did not generate better results with our data, doing so was shown in [9] to achieve better results on real data. Therefore we also show two different variations of the first protocol and a protocol based on secure two-party computation for matching with the weighted dice criterion with different privacy guarantees.

6.5.1 Private Set Intersection Based Protocols

Unweighted Dice Criterion

To compute the Dice criterion we need to know how many phone numbers appear in both input COIs; the *cardinality* of the intersection of the two COIs. We can use the Private Set Intersection protocol of Section 6.2.2 to compute the cardinality of the set intersection. But the Server S has to modify its behaviour in Step 2 b); instead of encoding y by computing $\mathrm{Enc}(rP(y)+y)$, S now encodes a "special" string Z, known to both parties, *i.e.*, S computes $\mathrm{Enc}(rP(y)+Z)$. In Step 3 of the protocol C now counts the number of ciphertexts that decrypt to the special string Z. The choice of $Z=0$ gives a computational efficiency improvement, as $\mathrm{Enc}(rP(y)+Z) = \mathrm{Enc}(rP(y))$.

In contrast to the original private set intersection protocol the Client now cannot learn the elements in the intersection.

Simple Weighted Dice Criterion

This private set intersection based protocol has lower privacy guarantees than the next one, but it is more efficient in computation and communication costs. It is a simple adaptation of the Private Set Intersection protocol of Section 6.2.2. In Step 2 b), instead of computing $\mathrm{Enc}(rP(y)+0)$ the server computes $\mathrm{Enc}(rP(y)+(y|w_y))$. Where $(y|w_y)$ stands for the concatenation of y and w_y. After decryption of all ciphertexts the client can determine the common elements with their respective weights and therefore compute the matching.

Obviously, in this protocol the client learns the elements of the intersection with their respective weights. But the efficiency is the same as that of the unweighted version.

Weighted Dice Criterion With Better Privacy Guarantees

In order to achieve better privacy, we split the sum in the numerator of the Dice criterion into two sums: $\sum_{o \in A \cap B} p_a(o) + p_b(o) = \sum_{o \in A \cap B} p_a(o) + \sum_{o \in A \cap B} p_b(o)$. We then compute the two sums by running the private set intersection protocol twice in opposite directions. In Step 2 b) the server now computes $\text{Enc}(rP(y) + (0|w_y))$. After decryption the client can sum the w_y to get $\sum_{o \in A \cap B} p_a(o)$.

Although the elements in the intersection are not transmitted, the parties might be able to extract information about the elements from the answers. As both parties learn the sum of their respective weights of the elements in the intersection, they might be able to infer which of their elements are contained in that sum and which are not. This issue becomes more difficult to ignore when multiple tests are made.

6.5.2 Secure Two-Party Computation Based Protocol

In order to achieve the highest privacy guarantees, we use garbled circuit techniques (see Section 2.3.3) to implement the unweighted Dice criterion. Note that the denominator can be computed by the Client without having to interact with the Server. The basic structure of the Boolean circuit to compute $\sum_{o \in A \cap B} w_{ao} + w_{bo}$ consists of k sub-circuits, which given a number / weight pair (n_i, w_i) of the client's COI and the full COI $((m_1, v_1), \ldots, (m_k, v_k))$ of the server, outputs either $w_i + v_j$ if $n_i = m_j$ or 0 otherwise. Figure 6.5.11 shows the design of such a sub-circuit. In the first layer of comparators (CMP) we compare n_i to all m_j. A comparator outputs 1 if the inputs match and 0 otherwise. The second layer consists of multiplexers (MUX) which, depending on the value of the control input, outputs either 0 or v_j. Note that at most only one multiplexer will output a value not equal to 0, because there can only be one comparator that detects the match $n_i = m_j$. The next layer consists of a chain of XOR gates and one multiplexer. The

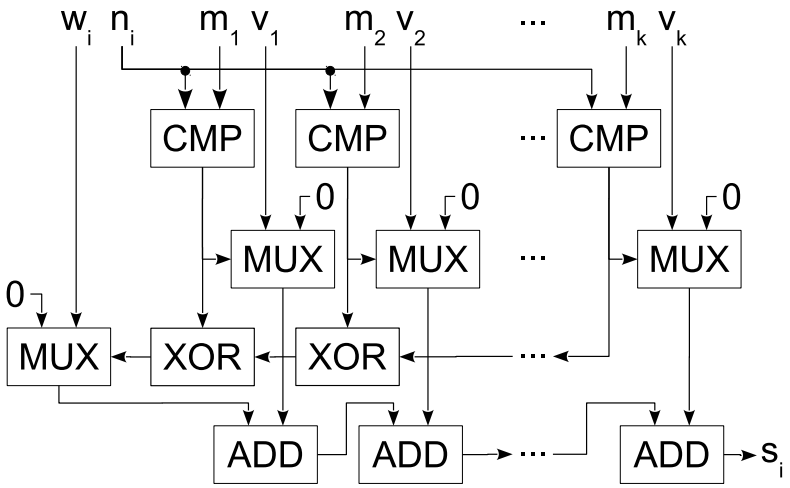


Figure 6.5.11: Part of the Boolean circuit to compute the unweighted Dice criterion. It outputs $s_i = w_i + v_j$ iff $n_i = m_j$ and $s_i = 0$ otherwise.

XOR chain adds all outputs of the comparators of layer one and that controls the multiplexer to output either w_i or 0. Again, there can only be one or zero comparators that output 1. Thus the multiplexer outputs w_i iff there is a match $n_i = m_j$. In the last layer we compute the sum of all outputs of the multiplexer. This is either $s_i = w_i + w_j$ or $s_i = 0$. At the end we have to compute the sum of the s_i which can be realised with a simple chain of ADD circuits.

For efficient constructions of circuits to compute CMP, MUX, and ADD see [88]. Overall we need

$$k^2\text{CMP} + (k^2 + k)\text{MUX} + (k^2 - k)\text{XOR} + (k^2 + k - 1)\text{ADD}$$

sub-circuits, with k denoting the number of top-k numbers in a COI.

The complexity of a circuit is measured in non-XOR gates, as the evaluation of XOR gates is essentially "for free" (see [90]). The complexity of these sub-circuits depends on the size of the inputs, *i.e.*, the number of bits needed to represent the inputs. Let l_n be the number of bits to represent a phone number and l_w be the number of bits to represent a weight. The

overall complexity, using the constructions of [88], is:

$$\text{Complexity} = 4k^2 l_n + (5k^2 + 5k - 4)l_w \text{ non-XOR gates.}$$

The parameter l_n is determined by the length of the telephone numbers and the consequences of the choice of k are examined in [72]. This leaves us with the question of how to choose l_w to achieve good detection results while keeping the complexity of the circuit low.

6.5.3 Quantised Weighted Dice Criterion

The weight of an element in a COI is represented by a real number. However, the privacy-preserving techniques we want to use in our protocols only work for a finite set of integers. Therefore we have to quantise the normalised weights, that is, we need to map the interval $[0, 1]$ to a set of integers that can be represented by b bits.

Although we could use the secure scaling protocol described in Chapter 4 to find a scaling factor for this mapping in a privacy-preserving manner, we refrain from doing so because the scaling factor in this application leaks no information. All inputs are from the interval $[0, 1]$ and the resolution does not have to be fine enough such that every input can be separated, as we are not interested in separating but in finding similarities. Thus we do not use secure scaling and save the extra overhead and instead use linear quantisation.

As seen in the previous section the complexity of the protocol scales linearly in the number of bits needed to represent the weight. In this section we investigate how the choice of bits for the weight affects the detection performance.

We use linear quantisation, that is, the interval $[0, 1]$ is divided into 2^b equidistant intervals $S_i = [\frac{i}{2^b}, \frac{i+1}{2^b}]$ for $0 \leq i < 2^b$. A weight is then quantised as $q(w) = \{\min i | w \leq \frac{i}{2^b}\}$.

To analyse the quality of the quantisation we ran the same test as for the classifier in Section 6.4.3 with the 90-day and the 1-day COIs. We use the weighted dice criterion with different levels of quantisation.

We determine the quality of the quantisation by comparing the classification result of the unquantised version with the quantised ones. Whenever the classification for a subscriber differs we count that as an error.

Figure 6.5.12 shows the errors for the different levels of quantisation. It shows that the numbers of errors decline with the increase of bits used

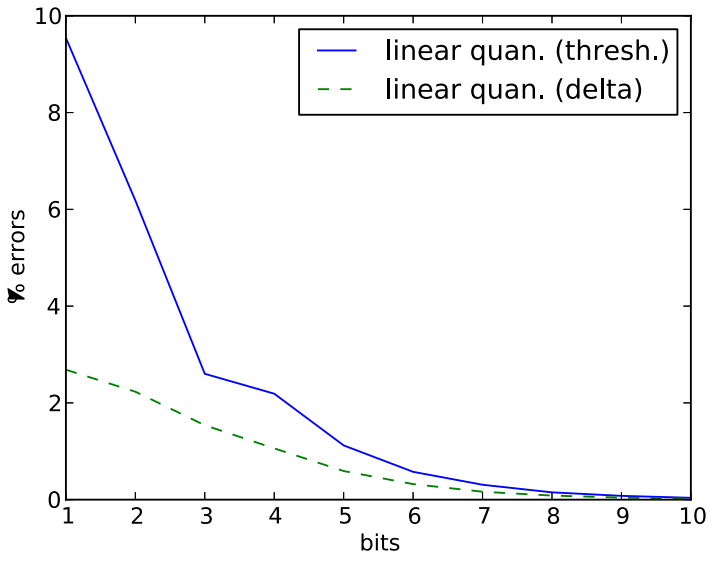


Figure 6.5.12: Comparison of different levels of the two quantisation types against no quantisation for threshold and delta classifier.

for quantisation. It also shows that the delta classifier is less affected by quantisation than the threshold classifier. From 10 bits onwards there is almost no difference between quantised and non-quantised classification.

We also test empirical quantisation, where we divided the interval $[0, 1]$ into 2^b sub-intervals, such that each interval contains the same quantity of weights in our dataset. However, this approach lead to worse results than the linear quantisation. We believe that the cause for this performance comes down to resolution of the quantisation around the maximum values, since

both classifiers take only either the highest or the two highest values into account. Our data is heavily skewed towards low values, so the empirical quantisation separates higher values rather poorly.

6.6 Implementation

We implemented the private set intersection protocol using the Python programming language. We chose Python because of its suitability for rapid prototyping and its vast collection of libraries. The core of the program is *Twisted* [2], an event-driven networking engine. It simplifies the complexity of networking by providing a suitable set of primitives.

The implementation consists of two programs, one implementing the client's, and one implementing the server's functionality. They communicate over the network using a TCP socket, thus they can be run on different computers.

Although the implementation is fully functional, we do not want to omit the limitations of our implementation. There is neither a graphical user interface nor a connector to a database. We also assume that the program is run through a secure communication channel. But since these are all fairly standard components we believe that they can be easily added to our implementation and their absence does not affect the measurements.

We use the same implementation of Paillier's homomorphic encryption scheme as in Section 5.5. It implements all the efficiency improvements described in Section 2.2.2.

For the evaluation of the garbled circuit based version of the protocol we used the secure two-party computation framework described in Chapter 3. It is amongst the most efficient frameworks and comes with all the tools necessary to create the Boolean circuit to compute the weighted Dice criterion.

6.6.1 Measurements

All measurements were run on a single iMac computer with a Core i3 3 Ghz dual core CPU to prevent network delays from distorting the results.

One-to-One Comparison

First we measure the computation and communication costs for the comparison of one COI with another COI.

Private Set Intersection We ran all measurements with a key size of 1024 bits. Thus, the message space is 1024 bits wide as well. Therefore we can easily encode a phone number and a high resolution quantised weight in one encryption. Table 6.6.1 shows the computation and communication costs for the different private set intersection based protocols.

Match. Crit.	Comp. Cost	Comm. Cost
Unweighted Dice	83 ms	5.25 KB
Simple Weighted Dice	113 ms	5.25 KB
Better Weighted Dice	113 ms	10.5 KB

Table 6.6.1: Comparison of the computation and communication costs for the different private set intersection based protocols.

The dominating factor for the computation costs are the encryptions with each encryption taking about 3 ms.

Weighted Dice with Secure two-Party Computation As shown in Section 6.5.2, the size of the Boolean circuit, and consequently the computation and communication costs, depend on the size of the inputs.

We generated Boolean circuits with 40 bit wide phone numbers (this allows 11 digits) and three different sized weights (6,8 and 10 bits). Table 6.6.2 shows the results.

Bitlength for weight (l_w)	Circuit Size (non-XOR gates)	Comp. Cost	Comm. Cost
6 bit	5214	18 ms	192.04 KB
8 bit	5652	19 ms	207.17 KB
10 bit	6090	21 ms	222.43 KB

Table 6.6.2: Comparison of the computation and communication costs for the garbled circuit based protocol

These measurements show that, apart from different privacy guarantees, the different protocols also have different strengths and weaknesses in terms of costs, depending on the underlying secure multiparty computation technique. The private set intersection based protocols take at least four times longer to do a comparison than the secure two-party computation based ones, but they have significantly less communication costs.

One-to-Many Comparison

The previous one-to-one comparison is useful but unrealistic, whereas in real applications many comparisons are needed. We therefore look at the costs for a one-to-many comparison. That is, the client can compare one COI with the server's database of n COIs. The naïve approach would be to run n one-to-one comparisons, but we can achieve better.

In the private set intersection based protocols, the client only has to encode its inputs as coefficients of a polynomial (Step 1) once. The Polynomial can be reused by the server for Step 2 without giving any advantage in obtaining information about the client's inputs.

The communication costs for a one-to-n comparison are: $\text{Costs}_{\text{comm.}} = 2.75 + 2.5n$ KB and the computation costs are: $\text{Costs}_{\text{comp.}} = 35 + 48n$ ms. Where the first summands are the costs for Step 1 of the protocol and the second summands are the costs for the n executions of Step 2.

The savings for the garbled circuit based protocol are not as pronounced since no part of the circuit can be reused. However, there are savings in establishing the initial configuration for the evaluation of the circuit. To obtain the right encryptions for its inputs the client has to run an oblivious transfer protocol. But those encryptions can be reused for all n comparisons and therefore the oblivious transfer has to be done only once.

For the 10 bit wide weights the communication costs are: $\text{Costs}_{\text{comm.}} = 196.47n + 5.37n + 20.58$ KB and the computation costs are: $\text{Costs}_{\text{comp.}} = 17n + n + 3$ ms. The first summands are the costs for evaluating the circuit, the second summands are the costs for exchanging the server's inputs and the last summands are the costs for exchanging the client's inputs.

Table 6.6.3 shows an estimation of the costs for running a 1-to-1000 and 1-to-100,000 comparison.

	1-to1000		1-to100,000	
Costs	PSI	S2PC	PSI	S2PC
computation	48 s	18 s	80 min	30 min
communication	2.44 MB	197 MB	244 MB	19.2 GB

Table 6.6.3: Estimation of the costs for 1-to-1000 and 1-to-100,000 comparisons for private set intersection (PSI) based and secure two-party computation (S2PC) based protocols.

Although the costs only grow linearly in the size of the database, they can be challenging for larger databases. However, as the comparisons are independent from each other, the execution can be parallelised. The database can be spread across several servers and Step 1 and Step 2 of the private set intersection protocol consists of k independent threads, which can be executed in parallel on k CPU cores.

6.7 Conclusion

In this chapter we proposed a model to generate synthetic phone records with nontrivial relationships. We used generated records to investigate how the choice of matching criteria and classifier affects the detection performance of the subscription fraud detection approach of [9]. We then extended their technique to several protocols that allow telcom operators to match their customers to the other telcom operators' fraudster database without revealing any additional information about the data than is necessary to classify the fraudster. Different levels of privacy are achieved by using two different secure multiparty computation techniques. We implemented these protocols to show feasibility and to compare their performance.

We believe that the benefit of co-operative fraud detection as proposed in this paper outweighs the costs for running the protocol, as telcom operators might only know a fraction of all fraudsters and therefore a combined fraudsters database promises better detection results. Fraud detection is an offline process and can be run periodically. Hence, we do not believe, that the slow execution time is critical. We provided several ideas to further improve our results, and realistically, telcom operators would likely implement a filter to only test suspicious subscribers.

We presented fraud detection solutions for two parties. If more than two parties want to co-operate, the resources needed to share fraudsters information will increase exponentially. However, as the underlying protocols for our solutions are extendable to the multi-party case, we consider this to be a potentially fruitful area of future research.

We believe that this signature-based detection approach might be useful in other scenarios like user tracking with browsing metadata or communication analysis on Email logs.

Chapter 7

Conclusion

Communication networks, like the Internet, have developed from one initial single entity controlled network into the network of networks of today. Each network is controlled by a different service provider. These providers have to engage in network management tasks like routing, traffic engineering, performance analysis and fraud detection. Some of these tasks require co-operation with other network providers, other tasks would strongly benefit from co-operation.

Although co-operation seems beneficial for the providers, they also compete, and much of the information they would need to share in order to co-operate is considered secret. In this thesis we investigated how secure multi-party techniques can enable co-operation without the need to reveal secret inputs to another competitor.

In the first half of this thesis we presented improvements to the practicability of secure multi-party techniques. We demonstrated a new implementation of Yao's two-party secure function evaluation protocol in Chapter 3, which leads to significantly better performance than previous implementations. The decoupling of memory demand from the circuit size, and an oblivious transfer extension with fixed memory consumption overhead, lead to a low memory footprint. This enables the evaluation of bigger circuits with less memory. With improved memory efficiency, and implementation

improvements like caching, we also achieved significantly faster execution times than previous frameworks. Further improvements such as decoupling of the unique gate ID from the circuit description, enables the framework to re-use parts of the circuits and enables dynamic circuit sizes.

This framework was subsequently used to implement the secure scaling protocol we presented in Chapter 4. It enables two parties to convert real-numbered privacy-preserving problems into the integer domain by scaling. Compared with previous solutions, we offer a more general approach that does not limit the choice of secure multiparty computation (SMC) techniques for the privacy-preserving problem, and does not introduce additional overhead. The main component of our solution is a novel algorithm for privacy-preserving random number generation.

In the second half of this thesis we demonstrated the application of SMC techniques to problems in network management. We presented STRIP, a distance-vector protocol, that allows routers to compute shortest paths without learning the distances of any paths. We also greatly reduce the spread of topology information by assuring that a participant only learns the next-hop router for a route. Previously proposed privacy-preserving routing protocols rely on a trusted third party for route computation. Our work maintains the distributed nature of a routing protocol as we do not rely on any additional players except a key distribution mechanism which already exists for BGP in the resource public key infrastructure framework RPKI. The basic privacy-preserving components of the protocol are easily extensible to implement other types of path metrics.

In Chapter 6 we proposed protocols for privacy-preserving fraud detection. They enable telecommunication providers to co-operate in detecting subscription fraud without violating protection laws for phone records. We extended a call-signature based subscription fraud detection approach to several protocols with different levels of privacy and two different SMC techniques which enable telecommunication providers to mine each others fraud-

ster databases. We also provide implementations of these protocols to show feasibility and to compare their performance.

Our approach throughout this thesis was pragmatic. By providing implementations of all our ideas we want to show that SMC-based protocols can be practical. We made the source code of all implementations available to encourage experimentations with and adaptation of our ideas. The results show that SMC could be highly beneficial to network operators.

Bibliography

[1] Simpy simulation package. `http://simpy.sourceforge.net/`.

[2] Twisted networking library. `http://twistedmatrix.com/trac/`.

[3] *FIPS 186-4: Digital Singature Standard (DSS)*. NIST, July 2013.

[4] Rakesh Agrawal and Ramakrishnan Srikant. Privacy-preserving data mining. In *Proceedings of the International Conference on Management of Data*, SIGMOD '00, pages 439–450, New York, USA, 2000. ACM.

[5] William Aiello, Yuval Ishai, and Omer Reingold. Priced oblivious transfer: How to sell digital goods. In *Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques: Advances in Cryptology*, EUROCRYPT '01, pages 119–135. Springer, 2001.

[6] Yonatan Aumann and Yehuda Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. In *Theory of Cryptography*, number 4392 in LNCS, pages 137–156. Springer, January 2007.

[7] A. Barbir, S. Murphy, and Y. Yang. Generic threats to routing protocols. RFC 4593, 2006.

[8] D. Beaver. Precomputing oblivious transfer. In *Advances in Cryptology – CRYPTO'95*, volume 963 of *LNCS*, pages 97–109. Springer, 1995.

[9] Richard A. Becker, Chris Volinsky, and Allan R. Wilks. Fraud detection in telecommunications: History and lessons learned. *Technometrics*, 52(1):20–33, 2010.

[10] M. Bellare and P. Rogaway. Optimal asymmetric encryption–how to encrypt with RSA. In *LNCS*, volume 950 of *Advances in Cryptology–EUROCRYPT'94*. Springer, 1994.

[11] A. Ben-David, N. Nisan, and B. Pinkas. FairplayMP: a system for secure multi-party computation. In *ACM Conference on Computer and Communications Security (CCS'08)*, pages 257–266. ACM, 2008.

[12] Rudolf Beran. Minimum hellinger distance estimates for parametric models. *The Annals of Statistics*, 5(3):445–463, 1977.

[13] Dimitri Bertsekas and Robert Gallager. *Data networks*. Prentice-Hall, Inc., NJ, USA, 1987.

[14] Tiziano Bianchi, Alessandro Piva, and Mauro Barni. On the implementation of the discrete fourier transform in the encrypted domain. *IEEE Transactions on Information Forensics and Security*, pages 86–97, March 2009.

[15] Anthony Bianco. *The Big Lie: Spying, Scandal, and Ethical Collapse at Hewlett Packard*. PublicAffairs, 2010.

[16] M. Blanton and M. Aliasgari. Secure computation of biometric matching. Technical Report CSE Technical Report 2009-03, University of Notre Dame, April 2009.

[17] M. Blanton and P. Gasti. Secure and efficient protocols for iris and fingerprint identification. In *European Symposium on Research in Computer Security (ESORICS'11)*, volume 6879 of *LNCS*, pages 190–209. Springer, 2011.

[18] Dan Bogdanov, Riivo Talviste, and Jan Willemson. Deploying secure multi-party computation for financial data analysis. In AngelosD. Keromytis, editor, *Financial Cryptography and Data Security*, volume 7397 of *LNCS*, pages 57–64. Springer, 2012.

[19] Peter Bogetoft, DanLund Christensen, Ivan Damgård, Ivan, Martin Geisler, Thomas Jakobsen, Mikkel Krigaard, JanusDam Nielsen, JesperBuus Nielsen, Kurt Nielsen, Jakob Pagter, Michael Schwartzbach, and Tomas Toft. Secure multiparty computation goes live. In Roger Dingledine and Philippe Golle, editors, *Financial Cryptography and Data Security*, volume 5628 of *LNCS*, pages 325–343. Springer, 2009.

[20] Beate Bollig and Ingo Wegener. Improving the variable ordering of obdds is np-complete. *IEEE Transactions on Computers*, 45(9):993–1002, 1996.

[21] Richard J. Bolton and David J. Hand. Statistical fraud detection: A review. *Statistical Science*, 17:235–255, 2002.

[22] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-dnf formulas on ciphertexts. In *Proceedings of the Second International Conference on Theory of Cryptography*, TCC'05, pages 325–341. Springer, 2005.

[23] Peter Boothe, James Hiebert, and Randy Bush. How Prevalent is Prefix Hijacking on the Internet? *NANOG 36*, February 2006.

[24] J. Boyar and R. Peralta. Concrete multiplicative complexity of symmetric functions. In *Mathematical Foundations of Computer Science (MFCS'06)*, volume 4162 of *LNCS*, pages 179–189. Springer, 2006.

[25] J. Boyar and R. Peralta. A new combinational logic minimization technique with applications to cryptology. In *Symposium on Experimental Algorithms (SOA'10)*, volume 6049 of *LNCS*, pages 178–189. Springer, 2010.

[26] J. Brickell and V. Shmatikov. Privacy-preserving graph algorithms in the semi-honest model. *Advances in Cryptology-ASIACRYPT 2005*, pages 236–252, 2005.

[27] R.E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, Aug 1986.

[28] S. Bugiel, S. Nürnberger, A.-R. Sadeghi, and T. Schneider. Twin Clouds: Secure cloud computing with low latency. In *Communications and Multimedia Security Conference (CMS'11)*, volume 7025 of *LNCS*, pages 32–44. Springer, 2011.

[29] Martin Burkhart, Mario Strasser, Dilip Many, and Xenofontas Dimitropoulos. SEPIA: Privacy-Preserving aggregation of Multi-Domain network events and statistics. In *Proceedings of the 19th USENIX Conference on Security*, 2010.

[30] Randy Bush, Olaf Maennel, Matthew Roughan, and Steve Uhlig. Internet optometry: assessing the broken glasses in Internet reachability. In *ACM SIGCOMM*, pages 242–253, 2009.

[31] Octavian Catrina and Amitabh Saxena. Secure computation with Fixed-Point numbers. In *Financial Cryptography and Data Security*, FC'10, pages 35–50. Springer, 2010.

[32] Communications Fraud Control Association CFCA. Global fraud loss survey 2013. `http://cfca.org/pdf/survey/CFCA2013GlobalFraudLossSurvey-pressrelease.pdf`, 2013.

[33] AC.-F. Chan. Symmetric-key homomorphic encryption for encrypted data processing. In *IEEE International Conference on Communications, ICC '09.*, pages 1–5, June 2009.

[34] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private information retrieval. *Journal of the ACM*, 45(6):965–981, November 1998.

[35] Josh Benaloh Clarkson. Dense probabilistic encryption. In *In Proceedings of the Workshop on Selected Areas of Cryptography*, pages 120–128, 1994.

[36] Mark Coates, Michael Rabbat, and Robert Nowak. Merging logical topologies using end-to-end measurements. In *ACM Internet Measurement Converence*, 2003.

[37] Chetan Sharma Consulting. US Market: Carrier Subscription Share Q3 2013. `http://image.slidesharecdn.com/uswirelessmarketq32013updatenov2013chetansharmaconsulting-131113221429-phpapp01/95/slide-12-638.jpg?cb=1384402693`.

[38] Corinna Cortes, Daryl Pregibon, and Chris Volinsky. Communities of interest. In *Advances in Intelligent Data Analysis*, pages 105–114. Springer, 2001.

[39] Corinna Cortes, Daryl Pregibon, and Chris Volinsky. Computational methods for dynamic graphs. *Journal of Computational and Graphical Statistics*, 12(4):950–970, December 2003.

[40] N. T. Courtois, D. Hulme, and T. Mourouzis. Solving circuit optimisation problems in cryptography and cryptanalysis. In *2nd IMA Conference Mathematics in Defence*, 2011.

[41] N. T. Courtois, D. Hulme, and T. Mourouzis. Solving circuit optimisation problems in cryptography and cryptanalysis. In *Special-purpose Hardware for Attacking Cryptographic Systems (SHARCS'12)*, pages 179–191, 2012.

[42] James Cowie. China's 18-minute mystery. *Renesys blog*, November 2010. http://www.renesys.com/blog/2010/11/chinas-18-minute-mystery.shtml.

[43] Ronald Cramer, Ivan Damgård, and Ueli Maurer. General secure multi-party computation from any linear secret-sharing scheme. In *Advances in Cryptology – EUROCRYPT*, LNCS, pages 316–334. Springer, 2000.

[44] Ivan Damgård and Mads Jurik. A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In Kwangjo Kim, editor, *Public Key Cryptography*, volume 1992, pages 119–136. Springer, 2001.

[45] Lee R. Dice. Measures of the amount of ecologic association between species. *Ecology*, 26(3):297–302, 1945.

[46] C. Ding, D. Pei, and A. Salomaa. *Chinese Remainder Theorem: Applications in Computing, Coding, Cryptography*. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 1996.

[47] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: the second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, volume 13 of *SSYM'04*, pages 303–320, San Diego, CA, 2004. USENIX Association.

[48] US Census Bureau Administration Division and Customer Services. US census bureau publications - statistical abstract 2011. http://www.census.gov/prod/www/statistical_abstract.html.

[49] Dyn Research. The new threat: Targeted internet traffic misdirection. http://research.dyn.com/2013/11/mitm-internet-hijacking/, 2013.

[50] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Proceedings of CRYPTO 84 on Advances in cryptology*, pages 10–18. Springer, 1985.

[51] D Evans, Y Huang, J Katz, and L Malka. Efficient privacy-preserving biometric identification. In *Proc. of Network and Distributed System Security Symposium, NDSS*, 2011.

[52] Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. *Commun. ACM*, 28(6):637–647, June 1985.

[53] FCC. Telecommunications act of 1996. `http://transition.fcc.gov/telecom.html`.

[54] Pierre-Alain Fouque, Jacques Stern, and Geert-Jan Wackers. Cryptocomputing with rationals. In *Financial Cryptography*, FC'02, pages 136–146. Springer, 2003.

[55] M. Franz, B. Deiseroth, K. Hamacher, S. Jha, S. Katzenbeisser, and H. Schröder. Secure computations on non-integer values. In *IEEE International Workshop on Information Forensics and Security (WIFS)*, pages 1–6, Dec 2010.

[56] M. J. Freedman, Y. Ishai, B. Pinkas, and O. Reingold. Keyword search and oblivious pseudorandom functions. In *Theory of Cryptography Conference (TCC'05)*, volume 3378 of *LNCS*, pages 303–324. Springer, 2005.

[57] MichaelJ. Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In *Advances in Cryptology - EUROCRYPT 2004*, pages 1–19. Springer, 2004.

[58] Keith Frikken. Secure multiparty computation. In *Algorithms and Theory of Computation Handbook, Second Edition*, pages 1–16. Chapman & Hall/CRC, 2009.

[59] William Gasarch. A survey on private information retrieval. *Bulletin of the EATCS*, 82:72–107, 2004.

[60] Craig Gentry. *A Fully Homomorphic Encryption Scheme*. PhD thesis, Stanford, CA, USA, 2009.

[61] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing*, STOC '09, pages 169–178, New York, NY, USA, 2009. ACM.

[62] Craig Gentry, Shai Halevi, and NigelP. Smart. Homomorphic evaluation of the aes circuit. In *Advances in Cryptology CRYPTO 2012*, volume 7417 of *LNCS*, pages 850–867. Springer, 2012.

[63] Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. A simple BGN-type cryptosystem from LWE. In *Proceedings of the 29th Annual International Conference on Theory and Applications of Cryptographic Techniques*, EUROCRYPT'10, pages 506–522. Springer, 2010.

[64] Oded Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, New York, NY, USA, 2004.

[65] Shafi Goldwasser and Silvio Micali. Probabilistic encryption & how to play mental poker keeping secret all partial information. In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, STOC '82, pages 365–377, New York, NY, USA, 1982. ACM.

[66] Vipul Goyal, Payman Mohassel, and Adam Smith. Efficient two party and multi party computation against covert adversaries. In *Advances*

in Cryptology – EUROCRYPT 2008, number 4965 in LNCS, pages 289–306. Springer, January 2008.

[67] T. Griffin and G. Huston. BGP wedgies. RFC 4264, 2005.

[68] T.G. Griffin and G. Wilfong. Analysis of the MED oscillation problem in BGP. In *ICNP*, pages 90–99, 2002.

[69] Henrik Grosskreutz, Benedikt Lemmen, and Stefan Rüping. Secure distributed subgroup discovery in horizontally partitioned data. *Transactions on Data Privacy*, 4(3):147–165, 2011.

[70] Debayan Gupta, Aaron Segal, Aurojit Panda, Gil Segev, Michael Schapira, Joan Feigenbaum, Jenifer Rexford, and Scott Shenker. A new approach to interdomain routing based on secure multi-party computation. In *ACM Workshop on Hot Topics in Networks*, HotNets-XI, pages 37–42, 2012.

[71] Wilko Henecka, Stefan Kögl, Ahmad-Reza Sadeghi, Thomas Schneider, and Immo Wehrenberg. Tasty: Tool for automating secure two-party computations. In *In ACM Conference on Computer and Communications Security (ACM CCS'10)*, pages 451–462, 2010.

[72] Shawndra B. Hill, Deepak K. Agarwal, Robert Bell, and Chris Volinsky. Building an effective representation for dynamic networks. *Journal of Computational and Graphical Statistics*, 15, 2006.

[73] Rahul Hiran, Niklas Carlsson, and Phillipa Gill. Characterizing large-scale routing anomalies: A case study of the china telecom incident. In *Passive and Active Measurement Conference*, 2013.

[74] A. Holzer, M. Franz, S. Katzenbeisser, and H. Veith. Secure two-party computations in ANSI C. In *ACM Conference on Computer and Communications Security (CCS'12)*, pages 772–783. ACM, 2012.

[75] Y. Huang, P. Chapman, and D. Evans. Privacy-preserving applications on smartphones. In *USENIX Workshop on Hot Topics in Security (HotSec'11)*, 2011.

[76] Y. Huang, P. Chapman, and D. Evans. Secure computation on mobile devices, 2011. Poster at IEEE Symposium on Security and Privacy.

[77] Y. Huang, L. Malka, D. Evans, and J. Katz. Efficient privacy-preserving biometric identification. In *Network and Distributed System Security (NDSS'11)*. The Internet Society, 2011.

[78] Yan Huang, David Evans, Jonathan Katz, and Lior Malka. Faster secure two-party computation using garbled circuits. In *Proceedings of the 20th USENIX Conference on Security*, SEC'11, pages 35–35, 2011.

[79] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In *Advances in Cryptology - CRYPTO 2003*, volume 2729 of *LNCS*, pages 145–161. Springer, 2003.

[80] Yuval Ishai and Anat Paskin. Evaluating branching programs on encrypted data. In *4th Theory of Cryptography Conference, TCC 2007, Amsterdam, The Netherlands*, volume 4392 of *LNCS*, pages 575–594. Springer, February 2007.

[81] S. Jarecki and X. Liu. Efficient oblivious pseudorandom function with applications to adaptive OT and secure computation of set intersection. In *Theory of Cryptography Conference (TCC'09)*, volume 5444 of *LNCS*, pages 577–594. Springer, 2009.

[82] K. Järvinen, V. Kolesnikov, A.-R. Sadeghi, and T. Schneider. Embedded SFE: Offloading server and network using hardware tokens. In *Financial Cryptography and Data Security (FC'10)*, volume 6052 of *LNCS*, pages 207–221. Springer, 2010.

[83] K. Järvinen, V. Kolesnikov, A.-R. Sadeghi, and T. Schneider. Garbled circuits for leakage-resilience: Hardware implementation and evaluation of one-time programs. In *Cryptographic Hardware and Embedded Systems (CHES'10)*, volume 6225 of *LNCS*, pages 383–397. Springer, 2010.

[84] Valentine Kabanets and Jin-Yi Cai. Circuit minimization problem. In *Proceedings of the Thirty-second Annual ACM Symposium on Theory of Computing*, STOC '00, pages 73–79, New York, NY, USA, 2000. ACM.

[85] A. A. Karatsuba and Y. Ofman. Multiplication of many-digital numbers by automatic computers. *SSSR Academy of Sciences*, 145:293–294, 1962.

[86] Simon Knight, Askar Jaboldinov, Olaf Maennel, Iain Phillips, and Matthew Roughan. Autonetkit: simplifying large scale, open-source network experimentation. *SIGCOMM Comput. Commun. Rev.*, 42(4):97–98, 2012.

[87] Donald E. Knuth. *The Art of Computer Programming*, volume 2. Addison-Wesley, 1998.

[88] V. Kolesnikov, A.-R. Sadeghi, and T. Schneider. Improved garbled circuit building blocks and applications to auctions and computing minima. In *Cryptology and Network Security (CANS'09)*, LNCS. Springer, 2009.

[89] Vladimir Kolesnikov, Ahmad-Reza Sadeghi, and Thomas Schneider. From dust to dawn: Practically efficient two-party secure function evaluation protocols and their modular design. *IACR Cryptology ePrint Archive*, page 79, 2010.

[90] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In *Automata, Languages and Programming*, number 5126 in LNCS, pages 486–498. Springer, January 2008.

[91] Benjamin Kreuter, Abhi Shelat, and Chih-Hao Shen. Billion-gate secure computation with malicious adversaries. In *Proceedings of the 21st USENIX Conference on Security Symposium*, Security'12, pages 14–14, Berkeley, CA, USA, 2012. USENIX Association.

[92] Louis Kruger, Somesh Jha, Eu-Jin Goh, and Dan Boneh. Secure function evaluation with ordered binary decision diagrams. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, CCS '06, pages 410–420. ACM, 2006.

[93] Craig Labovitz, Abha Ahuja, Abhijit Bose, and Farnam Jahanian. Delayed internet routing convergence. In *ACM SIGCOMM*, pages 175–187, 2000.

[94] M. Lepinski and S. Kent. An Infrastructure to Support Secure Internet Routing. RFC 6480, February 2012.

[95] Y. Lindell and B. Pinkas. Secure multiparty computation for privacy-preserving data mining. *Journal of Privacy and Confidentiality*, 1(1):59–98, 2009.

[96] Yehuda Lindell and Benny Pinkas. Privacy preserving data mining. In Mihir Bellare, editor, *Advances in Cryptology – CRYPTO 2000*, volume 1880 of *LNCS*, pages 36–54. Springer, 2000.

[97] Yehuda Lindell and Benny Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *Advances in Cryptology - EUROCRYPT 2007*, volume 4515, pages 52–78. Springer, 2007.

[98] Yehuda Lindell and Benny Pinkas. A proof of security of yao's protocol for two-party computation. *Journal of Cryptology*, 22(2):161–188, 2009.

[99] Ratul Mahajan, David Wetherall, and Thomas Anderson. Negotiation-based routing between neighboring ISPs. In *Proceedings of the 2nd Symposium on Networked Systems Design & Implementation*, volume 2 of *NSDI'05*, pages 29–42, Boston, MA, USA, 2005. USENIX Association.

[100] L. Malka. VMCrypt - modular software architecture for scalable secure computation. In *ACM Conference on Computer and Communications Security (CCS'11)*, pages 715–724. ACM, 2011.

[101] Dahlia Malkhi, Noam Nisan, Benny Pinkas, and Yaron Sella. Fairplay—a secure two-party computation system. In *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, SSYM'04, pages 20–20, 2004.

[102] Robert McMillan. Youtube outage underscores big Internet problem. *Infoworld*, 2008. `http://www.infoworld.com/t/applications/youtube-outage-underscores-big-internet-problem-702`.

[103] W. Melicher, S. Zahur, and D. Evans. An intermediate language for garbled circuits. Poster at IEEE Symposium on Security and Privacy, 2012.

[104] B. Mood, L. Letaw, and K. Butler. Memory-efficient garbled circuit generation for mobile devices. In *Financial Cryptography and Data Security (FC'12)*, LNCS. Springer, 2012.

[105] Wolfgang Mühlbauer, Anja Feldmann, Olaf Maennel, Matthew Roughan, and Steve Uhlig. Building an AS-topology model that captures route diversity. In *ACM SIGCOMM*, pages 195–206, 2006.

[106] David Naccache and Jacques Stern. A new public key cryptosystem based on higher residues. In *Proceedings of the 5th ACM Conference on Computer and Communications Security*, CCS '98, pages 59–66. ACM, 1998.

[107] Amit A. Nanavati, Siva Gurumurthy, Gautam Das, Dipanjan Chakraborty, Koustuv Dasgupta, Sougata Mukherjea, and Anupam Joshi. On the structural properties of massive telecom call graphs: Findings and implications. In *Proceedings of CIKM '06*, pages 435–444. ACM, 2006.

[108] Moni Naor and Benny Pinkas. Efficient oblivious transfer protocols. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '01, pages 448–457. Society for Industrial and Applied Mathematics, 2001.

[109] Moni Naor, Benny Pinkas, and Reuban Sumner. Privacy preserving auctions and mechanism design. In *Proceedings of the 1st ACM Conference on Electronic Commerce*, EC '99, pages 129–139. ACM, 1999.

[110] H.X. Nguyen and Matthew Roughan. Multi-Observer privacy preserving hidden markov models. In *Network Operations and Management Symposium (NOMS), 2012 IEEE*, pages 514–517, 2012.

[111] J. B. Nielsen, P. S. Nordholt, C. Orlandi, and S. S. Burra. A new approach to practical active-secure two-party computation. In *Advances in Cryptology – CRYPTO'12*, volume 7417 of *LNCS*, pages 681–700. Springer, 2012.

[112] Tatsuaki Okamoto and Shigenori Uchiyama. A new public-key cryptosystem as secure as factoring. In *Advances in Cryptology - EUROCRYPT '98*, volume 1403 of *LNCS*, pages 308–318. Springer, 1998.

[113] M. Osadchy, B. Pinkas, A. Jarrous, and B. Moskovich. SCiFI - a system for secure face identification. In *IEEE Symposium on Security and Privacy*, pages 239–254. IEEE, 2010.

[114] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *Advances in Cryptology – EUROCRYPT*, volume 1592, pages 223–238. Springer, 1999.

[115] Eric Parsonage, Hung X. Nguyen, and Matthew Roughan. Absorbing lexicographic products in metarouting. In *The 1st International Workshop on Rigorous Protocol Engineering (WRiPE)*, 2011.

[116] Iain Phillips, Olaf Maennel, Debbie Perouli, Rob Austein, Cristel Pelsser, Keiichi Shima, and Randy Bush. RPKI propagation emulation measurement: an early report. IETF Talk, July 2012.

[117] Clifton Phua, Vincent C. S. Lee, Kate Smith-Miles, and Ross W. Gayler. A comprehensive survey of data mining-based fraud detection research. *CoRR*, abs/1009.6119, 2010.

[118] Alex Pilosov and Tony Kapela. Stealing the Internet. In *Defcon 16*, 2008. `www.defcon.org/images/defcon-16/dc16-presentations/defcon-16-pilosov-kapela.pdf`.

[119] Benny Pinkas, Thomas Schneider, Nigel P. Smart, and Stephen C. Williams. Secure two-party computation is practical. In *Proceedings of the 15th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, ASIACRYPT '09, pages 250–267. Springer, 2009.

[120] Maurizio Pizzonia and Massimo Rimondini. Netkit: easy emulation of complex networks on inexpensive hardware. In *TridentCom*, pages 7:1–7:10, 2008.

[121] PRWeb. Telecom fraud survey. http://www.prweb.com/releases/neuraltechnologies/fraudandriskmanagement/prweb8472098.htm, May 2011.

[122] Michael O. Rabin. How to exchange secrets with oblivious transfer. *Technical Report TR-81*, 1981.

[123] Anirudh Ramachandran and Nick Feamster. Understanding the network-level behavior of spammers. In *ACM SIGCOMM*, pages 291–302, 2006.

[124] S. Ratnasamy and S. McCanne. Inference of multicast routing trees and bottleneck bandwidths using end-to-end measurements. In *IEEE INFOCOM*, volume 1, pages 353–360, 1999.

[125] Y. Rekhter and T. Li. A Border Gateway Protocol 4 (BGP-4). RFC 1654, March 1994.

[126] Fabio Ricciato and Martin Burkhart. Reduce to the max: A simple approach for Massive-Scale Privacy-Preserving collaborative network measurements (Short paper). In *Third International Workshop on Traffic Monitoring and Analysis (TMA)*, pages 100–107, Vienna, Austria, 2011.

[127] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, February 1978.

[128] Jeffrey Rott. Intel advanced encryption standard instructions (AES-NI). https://software.intel.com/en-us/articles/intel-advanced-encryption-standard-instructions-aes-ni/, 2012.

[129] M. Roughan and Y. Zhang. Privacy-preserving routing. Clean Slate Networks Workshop, Cambridge UK, September 2006.

[130] Matthew Roughan and Yin Zhang. Privacy-preserving performance measurements. In *Proceedings of the 2006 SIGCOMM workshop on Mining network data*, MineNet '06, pages 329–334, Pisa, Italy, 2006. ACM.

[131] Matthew Roughan and Yin Zhang. Secure distributed data-mining and its application to large-scale network measurements. *SIGCOMM Comput. Commun. Rev.*, 36:7–14, January 2006.

[132] Matthew Roughan and Yin Zhang. GATEway: symbiotic inter-domain traffic engineering. In *Proceedings of the 3rd International Conference on Performance Evaluation Methodologies and Tools*, ValueTools '08, pages 11:1–11:10, ICST, Brussels, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[133] Thomas Schneider. *Engineering Secure Two-Party Computation Protocols – Advances in Design, Optimization, and Applications of Efficient Secure Function Evaluation*. PhD thesis, Ruhr-University Bochum, Germany, February 2011.

[134] Iti Sharma. Fully homomorphic encryption scheme with symmetric keys. *CoRR*, abs/1310.2452, 2013.

[135] Samuel Shepard, Ray Kresman, and Larry Dunning. Data mining and collusion resistance. In *Proceedings of the World Congress on Engineering*, volume 1, pages 283–288, 2009.

[136] L. Subramanian, S. Agarwal, J. Rexford, and R.H. Katz. Characterizing the internet hierarchy from multiple vantage points. In *INFOCOM*, pages 618–627, 2002.

[137] J. Vaidya and C. Clifton. Privacy-preserving outlier detection. In *IEEE International Conference on Data Mining (ICDM '04)*, pages 233–240, Nov 2004.

[138] Kannan Varadhan, Ramesh Govindan, and Deborah Estrin. Persistent route oscillations in inter-domain routing. *Computer networks*, 32:1–16, 2000.

[139] J. von Neumann. Various techniques used in connection with random digits. monte carlo methods. *Nat. Bureau Standards*, 12:36–38, 1951.

[140] Feng Wang and Lixin Gao. On inferring and characterizing internet routing policies. In *ACM SIGCOMM*, pages 15–26, 2003.

[141] Yodai Watanabe, Junji Shikata, and Hideki Imai. Equivalence between semantic security and indistinguishability against chosen ciphertext attacks. In *Proceedings of the 6th International Workshop on Theory and Practice in Public Key Cryptography*, PKC '03, pages 71–84. Springer, 2003.

[142] Jianhong Xia and Lixin Gao. On the evaluation of AS relationship inferences. In *IEEE GLOBECOM*, pages 1373–1377, 2004.

[143] Andrew C Yao. Protocols for secure computations. *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, pages 160–164, 1982.

[144] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science, 1986*, pages 162–167. IEEE, October 1986.

[145] Yu Yu, J. Leiwo, and B. Premkumar. Securely utilizing external computing power. In *International Conference on Information Technology: Coding and Computing, ITCC 2005*, volume 1, pages 762–767, April 2005.

[146] Mingchen Zhao, Wenchao Zhou, Alexander J.T. Gurney, Andreas Haeberlen, Micah Sherr, and Boon Thau Loo. Private and verifiable interdomain routing decisions. In *ACM SIGCOMM*, pages 383–394, 2012.