Master thesis

A performance evaluation of secure comparison protocols in the two-party semi-honest model

Frank Blom
1822055

March 31, 2014

Abstract

Comparison protocols with secret inputs have been a main topic in multi-party computation research in the last few years due to their high complexity. Many different solutions involving homomorphic encryption, garbled circuits and secret sharing have been proposed, all claiming to be highly efficient of course. Unfortunately, a fair comparison between these solutions has yet to be made.

In this paper we present the reader with a fair comparison of state-of-the-art comparison protocols for the semi-honest two party setting. The performance analysis will be performed on the initialization, pre-computation and on-line phase of all considered protocols.

This paper will also give explanation and proof on many cryptographic and security concepts needed for performing a well educated evaluation.

The experimental results, together with the theoretic background collected in this paper are meant to give the reader a complete overview in the advantages and disadvantages of various proposed techniques. Both on the theoretic mathematical, as on the more practical implementation level.

Contents

Chapter 1

Introduction

Internet has become a vital part of our society. Whether we manage our bank accounts, do our taxes, surf the internet for entertainment or consider possible travel opportunities, our daily documented communication has become immense over the past decade. Consequently, an increased number of companies collect and harness this data in order to deliver personalized services through clever e-commerce algorithms. Take collaborative filtering to generate recommendations for instance [1]. Many of these algorithms rely on documented communication like browser history, location, preferences and much more. As attractive as these personalized services might seem, there is a catch. Data protection, often related to privacy issues, remains a big challenge within these systems [25]. Past experiences have shown that the regular protection protocols applied by companies are insufficient to uphold peoples privacy, as it often occurs that private information is left or lost in public places [8]. Furthermore, the notion that companies and agencies might sell, share or abuse confidential data is becoming a major concern [20].

The field of secure multi-party computation focuses on performing algorithms on encrypted or hidden, for example shared or garbled, data. Such techniques hold the key to a secure way of handling data without holding back the possibilities of algorithms running on this confidential data. In order to construct such secure signal processing algorithms, techniques like homomorphic encryption[2], garbled circuits [31] and secret sharing [27] are used. These systems allow for the service providers to hold encrypted privacy related data, perform algorithms on this data and consider customer service accordingly, but not access the actual protected data for they need not own the decryption keys.

This paper focuses on the search for an efficient way of performing secure comparison, one of the most used sub-protocols in secure multi-party computation. Because the comparison protocol is usually a sub-protocol encased in a much larger computation, we will assume that the inputs x and y, which need comparing, are encrypted $[\![x]\!], [\![y]\!]$ under a homomorphic scheme. Furthermore, we will assume that the result of the comparison performed, should be outputted as an encrypted value $[\![\delta]\!]$ as well. Where the bit δ is such that $\delta = 1$ if and only if x is strictly smaller than y. In order to keep our comparisons as realistic as possible, the maximum bit-length of x and y considered was chosen to be 25. We assume that only very few comparison have to be performed on values greater than $2^{25} - 1$ in practise.

We will evaluate state of the art comparison protocols involving homomorphic en-

cryption, garbled circuits and secret sharing. They will be evaluated on their round complexity, communication complexity, computation complexity and bandwidth in the two-party semi-honest security model. This evaluation will be performed by implementation and verification of all protocols on a single device.

Chapter 2

Preliminaries

In this chapter we will discuss some definitions and convenient notation which is very often used in Computer Science literature, but might be less familiar to the more mathematical reader. The basic principles used in the construction of these definitions are derived from [28].

Definition 2.0.1. We define the indefinite bit-length space

$$\{0,1\}^* := \cup_{n\in\mathbb{N}}\{0,1\}^n.$$

Note that any natural number $n \in \mathbb{N}$ can also be written as an element of $\{0,1\}^*$ by converting it to its bit-representation:

$$n = \sum_{i=0}^{k-1} 2^i n_i \quad \Rightarrow \quad n = (n_{k-1}, n_{k-2}, \ldots, n_1, n_0)_2.$$

We will sometimes implicitly switch between these two notations for positive integers without explicitly mentioning this. Now to make a choice between writing $n = 6$ as $(1,1,0)_2$ or $(0,0,0,1,1,0)_2$ or any other version of its binary representation, we introduce the following definition.

Definition 2.0.2. Let a strictly positive integer $x \in \{0,1\}^*$ be given. We then define the length $\ell(x)$ of x as the unique $k \in \mathbb{N}$ such that $x \in \{0,1\}^k$ and its most significant bit $x_{k-1} = \lfloor x/2^{k-1} \rfloor$ equals 1.
Also, we define $\ell(0) := 1$.

Note that the length of a natural number $n \in \mathbb{N}$ is the unique value given by

$$\ell(n) = \lfloor \log_2 n \rfloor + 1.$$

Also, for every odd prime p of length ℓ we have that $p = 2^\ell + q$ where $0 < q < 2^\ell$, which ensures that

$$\ell(p) = \lceil \log_2 p \rceil.$$

Definition 2.0.3. Let $x, y \in \{0,1\}^*$ be of length k and ℓ respectively. We then define the concatenation of x and y as follows:

$$x||y := (x_{k-1}, \ldots, x_0, y_{\ell-1}, \ldots, y_0)_2.$$

By using the concatenation notation we can ensure we write every element $x \in \{0,1\}^*$ in a unique way:

$$x = 1||(x_{\ell-2}, \ldots, x_0)_2,$$

where $\ell \in \mathbb{N}$ is the length of x.

Definition 2.0.4. We call a computation procedure for a computable function f an algorithm $\Pi : \{0,1\}^* \to \{0,1\}^*$ if Π on input x, denoted as $\Pi(x)$, outputs $f(x)$ by performing a finite sequence of instructions $I_0, I_1, \ldots, I_n$ for all x in the domain of f. Where each instruction is one of the following types:

- Arithmetic
  An arithmetic instruction " $a \leftarrow x \odot y$ " for input $x, y \in \{0,1\}^*$ where $\odot$ represents one of the following elementary (binary-)operations addition($+$), subtraction(-), multiplication($\times$) or integer division($\lfloor \cdot / \cdot \rfloor$).

- Branching
  A branching instruction " IF $x \diamond y$ GO TO i ", for input $x, y \in \{0,1\}^*$ and instruction index $i \leq n$.

- Halting
  The halting instruction " HALT " halts the execution of the program and outputs the value of the output parameter.

Definition 2.0.5. An algorithm Π of f is said to be of *polynomial time* if there exists a polynomial $p \in \mathbb{Z}[X]$ such that $\Pi(x)$ halts after performing at most $p(\ell(x))$ instructions for all x in the domain of f.

Definition 2.0.6. A polynomial time algorithm Π is said to be a *probabilistic polynomial time* (PPT) algorithm if it receives, in addition to its input x, a uniformly random element r with $\ell(r) \leq q(\ell(x))$ for some strictly positive valued polynomial $q \in \mathbb{Z}[X]$ $(q(n) > 0 \ \ \forall n \in \mathbb{N})$.

Definition 2.0.7. A computable function f is now considered *easy* if there exists a PPT algorithm Π which computes f. We call f *hard* if no such algorithm exists.

Chapter 3

Cryptographic Settings

In this chapter we introduce the reader to the principles and notation of different cryptographic settings. These settings will each give us different interpretations, tools and limitations to the general secure comparison topic.

3.1 Homomorphic Encryption

Consider the following scenario:
Alice $\mathcal{A}$ wants to enable Bob $\mathcal{B}$ to send her secret messages m belonging to some message space $\mathcal{M}$ without enabling Eve, the evil eavesdropper $\mathcal{E}$, to deduce m within reasonable time by eavesdropping on the conversation(s).

In order to do so, $\mathcal{A}$ releases some public key pk, a message space $\mathcal{M}$ and an (probabilistic) encryption scheme $E_{pk} : \mathcal{M} \to_R \mathcal{C}$, where $\mathcal{C}$ is called the cypher space. The secret key sk used in the decryption scheme $D_{sk} : \mathcal{C} \to \mathcal{M}$ is privately known to $\mathcal{A}$. That the encryption might be probabilistic means that some random component might be involved in the encryption protocol, i.e. it might be the case that for (fixed) message m and public key pk, encrypting m twice yields c_1 and c_2 successively, where $c_1 \neq c_2$. We do of course require $D_{sk}(c_1) = m = D_{sk}(c_2)$. In fact, the following should hold for any key pair (pk, sk) in order to be consistent:

$$D_{sk}\left(E_{pk}(m)\right) = m, \quad \forall m \in \mathcal{M}.$$

Definition 3.1.1. An encryption scheme with message space $\mathcal{M}$ and cypher space $\mathcal{C}$ is called *homomorphic* if there exist operations $\odot_C$ and $\odot_M$ such that for all $(m_1, m_2) \in \mathcal{M}^2$ and any consistent key pair (pk, sk), the following holds:

$$D_{sk}\left(E_{pk}(m_1) \odot_C E_{pk}(m_2)\right) = m_1 \odot_M m_2 \ .$$

3.1.1 Paillier

The Paillier encryption scheme was invented by Pascal Paillier. We will, in this example of a homomorphic encryption, give a detailed proof resembling the proofs published in the original paper[24].

First, some notation. Let p, q be two distinct odd primes of the same bit length such that $n := pq$ is a k-bit RSA modulus.

Lemma 3.1.1. *If $n = pq$ is the product of two distinct primes of the same bit length, then $gcd(n, \phi(n)) = 1$, where ϕ is the Euler's totient function.*

Proof. Note that under these assumptions $\gcd(n, \phi(n)) = \gcd(pq, (p-1)(q-1))$. without loss of generality we assume $q < p$. This means that $(q-1) < (p-1) < p$. Together with the fact that p is prime we have $\gcd(p, (p-1)(q-1)) = 1$. In order to prove the lemma we still need to show that $\gcd(q, (p-1)(q-1)) = 1$. We will do this by noting that $\gcd(q, (q-1)) = 1$ and proof that $\gcd(q, (p-1)) = 1$ also holds. To this end, assume the converse: $\gcd(q, (p-1)) \neq 1$. Because q is prime we have only one alternative, namely $\gcd(q, (p-1)) = q$. This means $(p-1)/q \geq 2$. But this contradicts our assumption that p and q are of the same size. Thus, $\gcd(q, (p-1)) = 1$. □

Let λ represents Carmichael's function applied to n:

$$\lambda := \lambda(n) = \operatorname{lcm}(\phi(p), \phi(q)) = \operatorname{lcm}(p-1, q-1).$$

Recall that the following property holds for all $w \in \mathbb{Z}_{n^2}^*$:

$$w^{\lambda n} \equiv 1 \pmod{n^2}.$$

Also, we have $\gcd(n, \lambda) = 1$, as a result of Lemma 3.1.1 combined with the notion that $\lambda(n)$ divides $\phi(n)$. We now define the integer-valued family of functions:

$$E_g : \mathbb{N}_{<n} \times \mathbb{Z}_n^* \to \mathbb{Z}_{n^2}^* \text{ for which } E_g(x, r) = g^x \cdot r^n \pmod{n^2},$$

where $g \in \mathbb{Z}_{n^2}^*$. In order to see that this function is well defined with respect to the equivalence classes $r \in \mathbb{Z}_n^*$, we should note that for all $k \in \mathbb{N}$ the following holds due to the binomial theorem:

$$(r + kn)^n = r^n + \binom{n}{1} knr^{n-1} + \mathcal{O}(n^2) = r^n + \mathcal{O}(n^2) \equiv r^n \pmod{n^2}.$$

In order to enforce a bit more structure on E_g we discuss our choice for g in the following theorem.

Theorem 3.1.1. *Define $\mathcal{B}_\alpha := \{r \in \mathbb{Z}_{n^2}^* : ord(r) = \alpha n\}$ for $\alpha \in \{1, \ldots, \lambda\}$. If g is an element of some $\mathcal{B}_\alpha$ then E_g is a bijection.*

Proof. Consider arbitrary $g \in \mathcal{B}_\alpha$ for some $\alpha \in \{1, \ldots, \lambda\}$. If E_g is injective then it is clearly surjective, because $|\mathbb{N}_{<n} \times \mathbb{Z}_n^*| = |\mathbb{Z}_{n^2}^*| = n(p-1)(q-1)$.

So we only need to prove that E_g is injective. Assume we have some pairs $(x, r_x), (y, r_y) \in \mathbb{N}_{<n} \times \mathbb{Z}_n^*$ such that $E_g(x, r_x) \equiv E_g(y, r_y) \bmod n^2$, then the following holds:

$$\begin{aligned}
& g^x r_x^n && \equiv && g^y r_y^n && \pmod{n^2} \\
\Rightarrow \quad & g^{x-y} r_x^n && \equiv && r_y^n && \pmod{n^2} \\
\Rightarrow \quad & g^{\lambda(x-y)} r_x^{\lambda n} && \equiv && r_y^{\lambda n} && \pmod{n^2} \\
\Rightarrow \quad & g^{\lambda(x-y)} && \equiv && 1 && \pmod{n^2}.
\end{aligned}$$

Hence $\alpha n | \lambda(x-y)$, which implies that $n | \lambda(x-y)$. Recall that $\gcd(\lambda, n) = 1$ and thus $n | (x-y)$. This means that $x - y \equiv 0 \pmod{n}$ which implies $x = y$ (recall that

$x, y < n$). Substituting this into the equation above yields $r_x^n \equiv r_y^n \pmod{n^2}$, which implies

$$\begin{aligned} & \left(r_x \cdot r_y^{-1}\right)^n \equiv 1 \pmod{n^2} \\ \Rightarrow \quad & r_x \cdot r_y^{-1} \equiv 1 \pmod{n} \\ \Rightarrow \quad & r_x^n \equiv r_y^n \pmod{n}. \end{aligned}$$

□

Definition 3.1.2. For $g \in \bigcup_{\alpha=1}^{\lambda} \mathcal{B}_\alpha$ and $w \in \mathbb{Z}_{n^2}^*$ we write $[\![w]\!]_g \in \mathbb{N}_{<n}$ as the unique (because E_g is bijective) element x for which there exists (unique) $r_x \in \mathbb{Z}_n^*$ such that

$$E_g(x, r_x) = w.$$

Lemma 3.1.2. *For all $g, h \in \bigcup_{\alpha=1}^{\lambda} \mathcal{B}_\alpha$:*

$$[\![w]\!]_h \cdot [\![h]\!]_g = [\![w]\!]_g.$$

Proof. By definition there exist r_x, r_y such that the following two equations hold:

$$h^x \cdot r_x^n \equiv w \pmod{n^2},$$

$$g^y \cdot r_y^n \equiv h \pmod{n^2},$$

where $x = [\![w]\!]_h$ and $y = [\![h]\!]_g$. Hence, by substitution:

$$w \equiv h^x \cdot r_x^n \equiv \left(g^y \cdot r_y^n\right)^x \cdot r_x^n = g^{xy} \cdot \left(r_y^x \cdot r_x\right)^n \pmod{n^2}.$$

□

Note that $[g]_g = 1$, hence from $[g]_h[h]_g = [g]_g$ we see that $[h]_g = [g]_h^{-1}$ for all $g, h \in \bigcup_{\alpha=1}^{\lambda} \mathcal{B}_\alpha$. Now let $L_n(u) := (u-1)/n$ be an integer-valued function which is clearly well defined for all $u \in \mathcal{S}_n = \{u < n^2 \ : \ u = 1 \mod n\}$. Then the following lemma will give us some insight in the possible uses of L_n.

Lemma 3.1.3. $L_n(w^\lambda \bmod n^2) = \lambda [\![w]\!]_{n+1} \bmod n$, *for all $w \in \mathbb{Z}_{n^2}^*$.*

Proof. Consider arbitrary $w \in \mathbb{Z}_{n^2}^*$. Note that $1 + n \in \mathcal{B}_1$, because $(n+1)^z = 1 + zn + \mathcal{O}(n^2)$ by the binomial theorem. So by Theorem 3.1.1 there exists a unique pair $(x, r_x) \in \mathbb{N}_{<n} \times \mathbb{Z}_n^*$ such that $w \equiv (1+n)^x \cdot r_x^n \pmod{n^2}$; recall $x = [\![w]\!]_{n+1}$. Then also:

$$w^\lambda \equiv (1+n)^{\lambda x} \cdot r_x^{\lambda n} \equiv (1+n)^{\lambda x} \equiv 1 + \lambda x n \pmod{n^2}.$$

Hence

$$L_n(w^\lambda \mod n^2) = L_n(1 + \lambda x n) = \frac{1 + \lambda x n - 1}{n} = \lambda x = \lambda [\![w]\!]_{n+1} \pmod{n}.$$

□

Now that we have build all the tools, we can build the actual homomorphic encryption scheme. Let $\mathcal{A}$ pick the large primes p and q and set $n = pq$. She then randomly select a base $g \in \bigcup_{\alpha=1}^{\lambda} \mathcal{B}_\alpha$ by selecting a random $g \in \mathbb{Z}_{n^2}^*$ and checking whether $\gcd\left(L_n(g^\lambda \mod n^2), n\right) = 1$. She then sets $\mu = L_n(g^\lambda \mod n^2)^{-1} \bmod n$ and releases the public key $pk = (n, g)$. The secret key $sk = (\lambda, \mu)$ she keeps to herself for decryption.

- Encryption:
  Let m be a message in $\mathbb{N}_{<n}$. Bob then picks a random $r \in \mathbb{Z}_n^*$ en encrypts:
$$c = E_g(m, r) = g^m \cdot r^n \pmod{n^2}.$$

- Decryption:
  Let c be a received cypher, then we decrypt as follows:
$$m = \mu L_n(c^\lambda \mod n^2) \pmod{n}.$$

Correctness of the decryption protocol follows almost immediately from Lemma 3.1.2 and 3.1.3:

$$\begin{aligned} \mu L_n(c^\lambda \mod n^2) &= L_n(g^\lambda \mod n^2)^{-1} \cdot L(c^\lambda \mod n^2) \\ &= \lambda^{-1}[\![g]\!]_{n+1}^{-1} \cdot \lambda[\![c]\!]_{n+1} = [\![c]\!]_{n+1} \cdot [\![n+1]\!]_g = [\![c]\!]_g = m \pmod{n}. \end{aligned}$$

Note that this encryption scheme is indeed homomorphic:

$$E_g(m_1, r_1) \cdot E_g(m_2, r_2) = g^{m_1} \cdot r_1^n \cdot g^{m_2} \cdot r_2^n = g^{m_1+m_2} \cdot (r_1 r_2)^n,$$

which decrypts to $m_1 + m_2 \bmod n$. Consequently, decryption of $E_g(m, r)^c$ yields $c \cdot m$ for scalar $c \in \mathbb{Z}$. Such a homomorphic encryption scheme is called additive.

A 'short-cut' version of the Paillier scheme can be implemented by setting $g = n + 1 \in \mathcal{B}_1$. Then by the binomial theorem we have

$$E_g(m, r) = g^m \cdot r^n = g^m \cdot r^n = (n+1)^m \cdot r^n \equiv (mn+1) \cdot r^n \pmod{n^2}.$$

Decryption can be done efficiently by applying the Chinese remainder Theorem: We split $L_n(x)$ in two by defining $L_p(x) = (x-1)/p$ and $L_q(x) = (x-1)/q$ defined over the p and q-Sylow subgroups of $\mathbb{Z}_{p^2}^*$ and $\mathbb{Z}_{q^2}^*$ respectively:

$$\mathcal{S}_p = \{x \in \mathbb{Z}_{p^2}^* \ : \ x \equiv 1 \bmod p\},$$

$$\mathcal{S}_q = \{x \in \mathbb{Z}_{q^2}^* \ : \ x \equiv 1 \bmod q\}.$$

In order to make this decryption protocol as fast as possible we pre-compute

$$\mu_p = L_p(g^{p-1} \bmod p^2)^{-1} \bmod p,$$

$$\mu_q = L_q(g^{q-1} \bmod q^2)^{-1} \bmod q,$$

$$p^{-1} \bmod q$$

and

$$q^{-1} \bmod p.$$

Now we decrypt by computing

$$m_p = \mu_p L_p(c^{p-1} \bmod p^2) \mod p,$$

$$m_q = \mu_q L_q(c^{q-1} \bmod q^2) \mod q.$$

Then by the Chinese remainder theorem we have:

$$m = q(q^{-1} \bmod p)m_p + p(p^{-1} \bmod q)m_q \pmod{n}.$$

3.1.2 DGK

The cryptographic system invented by Ivan Damgård, Martin Geisler and Mikkel Krøigaard [15] can be efficiently used to determine whether a cypher c corresponds with message $m = 0$. This construction is particularly helpful for implementing secure comparison. Why this is helpful is further explained in Subsection 5.1. The original construction of the protocol was corrected due to a security flaw observed by Claudio Orlandi [11].

Let $\ell < t < k$ be the chosen security parameters. Choose a k-bit RSA modulus $n = pq$ such that there exist primes u (ℓ-bit), v_p and v_q (both t-bit) for which the following hold:

$$u|(p-1), \quad u|(q-1), \quad v_p|(p-1) \quad \text{and} \quad v_q|(q-1).$$

Fix such u, v_p, v_q and set $v = v_p \cdot v_q$. Choose (randomly) two elements $g, h \in \mathbb{Z}_n^*$ with $ord_n(g) = uv$, $ord_p(h) = v_p$ and $ord_q(h) = v_q$. We consider public key $pk = (n, g, h, u)$ and secret key $sk = (p, q, v_p, v_q)$.

In order to encrypt a message $m \in \mathbb{N}_{<u}$, we use the following encryption function:

$$E_{pk}(m, r) = g^m \cdot h^r \pmod{n},$$

where r a uniformly random integer with at least $\frac{5}{2}t$ bits. Note that r needs considerably more bits than v to ensure that h^r is statistically indistinguishable[1] from a uniformly random element in $\langle h \rangle$, the (multiplicative) group generated by h.

Determining whether c encrypts a zero is very easy.

Lemma 3.1.4. *$c^v \equiv 1 \bmod n$ if and only if c encrypts 0.*

Proof. Consider some consistent key pair (pk, sk) and let $c = E_{pk}(m, r)$ be some encryption of $m < u$. $h^v \equiv 1$ in both $\mathbb{Z}_p^*$ and $\mathbb{Z}_q^*$, hence also $h^v \equiv 1 \pmod{n}$. Which gives us:

$$c^v = (g^m \cdot h^r)^v = g^{mv} \cdot (h^v)^r \equiv g^{mv} \pmod{n}.$$

So $c^v \equiv 1 \bmod n$ if and only if $g^{mv} \equiv 1 \bmod n$. But from the set-up of the encryption we know that g has order uv modulo n and $m < u$. Thus $c^v \equiv 1 \bmod n$ if and only if $m = 0$ as required. □

For proper decryption we can do no better than to simply brute force the message space by means of v_p.

3.1.3 Okamoto-Uchiyama

In Section 3.2 we introduce the notion of Oblivious Transfer (OT). Such OT-schemes desire a cryptographic construction similar to that of Paillier, but with a smaller message space. Such a homomorphic encryption was originally presented by Okamoto and Uchiyama [23] and improved by Coron, Naccache and Paillier [9].

Let $\mathcal{S}_p$ be the p-Sylow subgroup of $\mathbb{Z}_{p^2}^*$ for some prime p:

$$\mathcal{S}_p = \{x \in \mathbb{Z}_{p^2}^* \ : \ x \equiv 1 \bmod p\}.$$

Then we let $L_p(x) := (x-1)/p$ be the logarithm function over $\mathcal{S}_p$.

[1] For a formal formulation of (statistical) indistinguishability, see Definition 4.1.2.

Theorem 3.1.2. *$L_p : \mathcal{S}_p \to \mathbb{Z}_p$ is an isomorphism.*

Proof. Remember that the order of a p-Sylow subgroup P of G can be found from the order of G. Write $|G| = p^n m$ where p does not divide m, then $|P| = p^n$. We have $|\mathbb{Z}_{p^2}^*| = \phi(p^2) = p(p-1)$, So $|\mathcal{S}_p| = p = |\mathbb{Z}_p|$. Hence, if L_p is injective then it must be bijective.

So assume we have $x, y \in \mathcal{S}_p$ for which $L_p(x) \equiv L_p(y) \bmod p$. Then we write $x \equiv ap + 1$ and $y \equiv bp + 1$ modulo p^2, where $a, b < p$. Hence

$$L_p(x) = \frac{x-1}{p} \equiv a \mod p,$$

$$L_p(y) = \frac{y-1}{p} \equiv b \mod p.$$

Now, because a and b were both smaller than p we see that $L_p(x) \equiv L_p(y) \bmod p$ implies $a = b$ which in turn implies that $x \equiv y \bmod p^2$. So $L_p(x)$ is indeed bijective.

Now assume we have $x, y \in \mathcal{S}_p$. Then

$$L_p(xy) = \frac{xy-1}{p} = \frac{(x-1)(y-1) + (x-1) + (y-1)}{p} = L_p(x)(y-1) + L_p(x) + L_p(y).$$

And because $(y-1) \equiv 0 \bmod p$, we see that

$$L_p(xy) \equiv L_p(x) + L_p(y) \pmod p.$$

So we conclude that $L_p : \mathcal{S}_p \to \mathbb{Z}_p$ is an isomorphism. □

From the fact that L_p is an isomorphism between $\mathcal{S}_p$ and $\mathbb{Z}_p$, the following lemma follows almost immediately.

Lemma 3.1.5. *If $x \in \mathcal{S}_p$ such that $L_p(x) \not\equiv 0 \bmod p$. then for all $m \in \mathbb{Z}_p$ and $y = x^m \bmod p^2$, we have*

$$m = \frac{L_p(y)}{L_p(x)} \pmod p.$$

Proof.

$$\frac{L_p(y)}{L_p(x)} = \frac{L_p(x^m)}{L_p(x)} = \frac{mL_p(x)}{L_p(x)} = m \pmod p.$$

□

Now that we have the building blocks, let's define our keys, encryption and decryption. Pick two primes of length p, q of the same bit-length such that $3\ell(p) > k$, where k is the desired length of an RSA modulus. Also, let $p - 1 = tu$, where t is a large (say 224-bit) prime. We set $n = p^2 q$ and select random $g, g' < n$ such that g^{p-1} has order p in $\mathbb{Z}_{p^2}^*$. Set $G := g^u \bmod n$ and $H := g'^{nu} \bmod n$. Then the public key and private key are given by $pk = (n, G, H)$ and $sk = (p, q)$. A message $m \in \mathbb{N}_{<p}$ is encrypted by

$$E_{pk}(m, r) = G^m \cdot H^r \pmod n,$$

where $r \in \mathbb{N}_{<n}$ is a uniform random number. The decryption of a given cypher text c is performed in two steps, first we calculate

$$c^t = g^{m(p-1)} g'^{nr(p-1)} = \left(g^{p-1}\right)^m \pmod{p^2}.$$

Then we apply Lemma 3.1.5 in order to receive

$$m = \frac{L_p(c^t)}{L_p(g^{p-1} \bmod p^2)}.$$

Correctness follows from the fact that $g^{p-1} \in \mathcal{S}_p$ by Fermat's little theorem.

3.2 Garbled Circuits

As it is currently a standard approach for secure two-party computation we consider garbled cicuits (GC) originating from [31].

The literature generally assumes two parties in a Garbled Circuit protocol namely a *constructor*, who builds the circuit and a *evaluator*, who performs a circuit evaluation with garbled inputs. In our setting we let Alice be the constructor and Bob be the evaluator. The goal is to let Alice and Bob verify the output of some given function along a (binary) circuit without giving away the values of their private inputs. In order to do so, Alice creates a garbled version of the circuit as described in Subsection 3.2.1. The GC creation method allows for "free" XOR-gates due to the work of [18]. This is done by selecting two garbled values w_i^0, w_i^1 in a clever way for every (i'th) wire in the circuit. Note that w_i^b does not reveal b. Alice also creates garbled tables T_i for every non XOR-gate. These tables contain the garbled outputs associated with specific inputs to gate i. We only consider circuits which allow for at most two binary inputs and output one bit correspondingly, because every computable function can be modelled by a Turing machine which can very naturally be transformed into a binary circuit. Also we assume that all gates $G_1, \ldots, G_k$ are ordered such that no gate G_i takes input which is output from a successive gate G_j $(j > i)$.

After Alice has generated the garbled circuit, she sends the circuit along with her garbled inputs $\{w_i^{x_i}\}_{i=0}^{\ell-1}$ to Bob, where $x = \sum_{i=0}^{\ell-1} 2^i x_i$ is her true input. It is now Bob's job to evaluate the circuit with the garbled inputs received from Alice as well as his own inputs, this procedure is described in Subsection 3.2.3. Unfortunately the client does not know the garbled values corresponding to his inputs. For that matter we introduce a precomputed oblivious transfer in Subsection 3.2.2.

3.2.1 Generating the Garbled Circuit

The garbled circuit construction we consider comes from [18]. This construction was specifically designed for the two party semi-honest model as described in Chapter 4. This construction allows for computation "free" XOR gates by introducing a global offset Δ which ensures that no garbled tables are needed for these gates and evaluation is done via a simple bitwise XOR of the garbled values.

For the construction of the circuit, we let N be the security parameter. each garbled value w_i^b will then consist of a key $k_i^b \in \{0,1\}^N$ and a permutation bit

$p \in \{0,1\}$. We let $H : \{0,1\}^* \to \{0,1\}^{N+1}$ be some random Oracle[2].

Alice's construction of the garbled circuit

1. Pick random offset $\Delta \in \{0,1\}^N$.
2. For each circuit input wire W_i we garble the input value:
   a) Choose random key and permutation bit $k_i^0 \in \{0,1\}^N$, $p_i^0 \in \{0,1\}$ and let the first wire value be given by the ordered pair $w_i^0 = (k_i^0, p_i^0)$.
   b) Set $w_i^1 = (k_i^1, p_i^1) = (k_i^0 \oplus \Delta, p_i^1 \oplus 1)$.
3. For each gate G_i with output wire W_c and input wires W_a and W_b we garble the output wire values:
   a) If G_i is an XOR:
      (i) We set $w_c^0 = (k_c^0, p_c^0) = (k_a^0 \oplus k_b^0, p_a^0 \oplus p_b^0)$,
      (ii) and $w_c^1 = (k_c^1, p_c^1) = (w_c^0 \oplus \Delta, p_c^0 \oplus 1)$.
   b) If G_i is not an XOR:
      (i) Choose random key and permutation bit $k_c^0 \in \{0,1\}^N$, $p_c^0 \in \{0,1\}$ and let the first output wire value be given by the ordered pair $w_c^0 = (k_c^0, p_c^0)$.
      (ii) Set $w_c^1 = (k_c^1, p_c^1) = (k_c^0 \oplus \Delta, p_c^1 \oplus 1)$.
      (iii) Create garbled table

$$T_i = \begin{bmatrix} e_{0,0} & e_{0,1} \\ e_{1,0} & e_{1,1} \end{bmatrix},$$

      by computing

$$e_{p_a^{v_a}, p_b^{v_b}} = H(k_a^{v_a} || k_b^{v_b} || i) \oplus \left(k_c^{G_i(v_a,v_b)} || p_c^{G_i(v_a,v_b)}\right),$$

      for all $(v_a, v_b) \in \{0,1\}^2$.
4. For each circuit output wire W_i we create a garbled output table:

$$T_{out} = \begin{bmatrix} e_0 & e_1 \end{bmatrix}$$

   by computing

$$e_{p^v} = H(k_i^v || \text{“}out\text{”} || i) \oplus v$$

   for both $v = 0$ and $v = 1$.

[2]In practise, H can be implemented by means of a hash function.

A few remarks about this construction are in order. First off, we see that there are no collisions in the tables because $p_i^0 = p_i^1 \oplus 1$. Secondly, to prevent further collision in random Oracle H we define the "out" parameter. Which is a fixed N bit value. Also, we added the index i of the corresponding gate G_i for the purpose of non-collision.

3.2.2 Oblivious Transfer

When a circuit is constructed the garbled values, to which the input has to be converted, are fixed. It is clear that Alice can send her garbled input corresponding to her construction of the garbled circuit together with the garbled tables to Bob. The question remains, how should Bob know how to garble his input accordingly without revealing his private input tot Alice? The answer lies in Oblivious transfer.

Definition 3.2.1. We call a two party protocol a "Chosen 1-out-of-2 Oblivious Transfer", notation: $\binom{2}{1}$-OT, when the following criteria are met:

1. The protocol takes a single input bit $b \in \{0, 1\}$ from Party 1.
2. Party 2 holds two secret values s_0, s_1.
3. The protocol outputs s_b to Party 1 securely, i.e. the value of s_{1-b} and b are securely hidden from Party 1 and Party 2 respectively within the used security model.

In our case we will focus on Chosen 1-out-of-2 Oblivious Transfer which is secure in the Semi-Honest model. For a formal definition on two-party semi-honest security, we refer the reader to Chapter 4.

There are many variation of oblivious transfer. So choices have to be made. We choose to consider the following $\binom{2}{1}$-OT protocol as described by Moni Naor and Benny Pinkas [21]. Which is an amended version of the protocol by Belli-Micali [6], by introducing a random Oracle primitive.

A basic Chosen 1-out-of-2 Oblivious Transfer:

This protocol acts over a sub-group $G_q \subseteq \mathbb{Z}_q^*$ of prime order, where $q|(p-1)$ and p is prime. We then let g be a generator for G_q, i.e. $\langle g \rangle = G_g$. To ensure security for this protocol we assume that the computational Diffie-Hellman assumption holds for this generator g. Let H represent a random Oracle, which can in practise be implemented by means of a hash function.

1. Bob picks a random constant $C \in G_q$.
2. Bob picks a random integer $1 \leq k \leq q$ and sets two public keys:

$$PK_b = g^k, \quad PK_{1-b} = C/PK_b.$$

3. Bob sends PK_0 to Alice.

4. Alice computes $PK_1 = C/PK_0$ and picks two randomizers $1 \leq r_0, r_1 \leq q$.

5. Alice computes

$$x_0 = H(PK_0^{r_0}), \quad x_1 = H(PK_1^{r_1}),$$
$$e_0 = x_0 \oplus s_0, \quad e_1 = x_1 \oplus s_1,$$
$$g_0 = g^{r_0}, \quad g_1 = g^{r_1},$$

   and sends (e_0, e_1, g_0, g_1) to Bob.

6. Bob computes $x_b = H(PK_b^{r_b}) = H\left(g_b^k\right)$ to retrieve

$$s_b = e_b \oplus x_b.$$

Now that we have the tools to preform a basic $\binom{2}{1}$-OT protocol, we can think of this protocol as the abstract concept of a primitive $\binom{2}{1}OT$ which on input (s_1, s_2, b) outputs (b, s_b). In this notation the $\binom{2}{1}$-OT protocol can be described in the following abstract way.

A	input $s_0,\ s_1 \in \Omega$
B	input $b \in \{0, 1\}$
$A \to \binom{2}{1}OT$	(s_0, s_1)
$B \to \binom{2}{1}OT$	b
$B \leftarrow \binom{2}{1}OT$	(b, s_b)

Where A and B denote the abstract representations of Alice and Bob respectively, the symbol "$\to$" represents the abstraction of information transference and Ω is some finite set which contains the secrets. As opposed to the non-constructive proof given in [5] we can proof the precomputed self-reduction of $\binom{2}{1}OT$ in a constructive manner.

Theorem 3.2.1. *Given input s_0, s_1 at the start of a "on-line" phase, we can reduce the basic Chosen 1-out-of-2 Oblivious Transfer to a precomputed variant of itself, notation:*

$$\binom{2}{1}OT \overset{pre}{\leq} \binom{2}{1}OT$$

Proof. Let $x \leftarrow_R X$ denote the action of picking a uniformly random element from a finite set X. Then the pre-computation and on-line phase can preformed as follows

Precomputation :		*Online* :	
A	$r_0 \leftarrow_R \Omega,\ r_1 \leftarrow_R \Omega$	A	input $s_0,\ s_1 \in \Omega$
B	$c \leftarrow_R \{0, 1\}$	B	input $b \in \{0, 1\}$
$A \to \binom{2}{1}OT$	(r_0, r_1)	$B \to A$	$e = b \oplus c$
$B \to \binom{2}{1}OT$	c	$A \to B$	$(x_0, x_1) = (s_0 \oplus r_e, s_1 \oplus r_{1-e})$
$B \leftarrow \binom{2}{1}OT$	(c, r_c)	B	output $s_b = x_b \oplus r_c$

Note that we have successfully performed a precomputed self-reduction of the basic Chosen 1-out-of-2 Oblivious Transfer, because all assertions to the $\binom{2}{1}OT$ primitive are preformed in the pre-computation. □

The constructive nature of this theorem's proof enables us to shift the computationally exhaustive part to the pre-computation without to much effort.

While implementation of a $\binom{2}{1}$-OT seems easy at first, there is one major pitfall still. In practise we will need to find a prime q such that it divides $p-1$ where p is also prime. Furthermore, we need to find a generator g for G_q for which the computational Diffie-Hellman assumption holds. This last statement implies that $p-1$ should have a large prime factor. So the idea is, lets make q a very large prime factor of $p-1$. specifically, we can let q be a Sophie Germain prime, which makes finding a generator very easy.

Definition 3.2.2. A prime q is called a *Sophie Germain prime* if $p = 2q+1$ is also prime. p is then called a *safe prime*.

Note that when q is Sophie Germain, than indeed $q|p-1$. And in order to find a generator g we can pick random elements x of $\mathbb{Z}_p^*$ for which

$$x^2 \not\equiv 1 \pmod{p^2},$$

$$x^q \not\equiv 1 \pmod{p^2},$$

and set $g = x^2$. This construction results in

$$\langle g \rangle = \{x^2 \bmod p^2, x^{2\cdot 2} \bmod p^2, x^{2\cdot 3} \bmod p^2, \ldots, x^{2\cdot q} = x^{p-1} \equiv 1 \bmod p^2\} = G_q,$$

which ensures $|G_q| = q$. The only drawback about this method is that generating a Sophie Germain prime tends to be extremely costly in terms of computation complexity[3]. Fortunately, nothing withholds us from taking a fixed number. The Sophie Germain prime used in our implementation can be found in Appendix A.1

3.2.3 Evaluation

After receiving all garbled inputs (k_i, p_i) and garbled tables T_i, T_{out}, Bob can evaluate the inputs over the circuit by performing the following steps.

Bob's evaluation of the garbled circuit

1. For each circuit input wire W_i, set its value to the corresponding garbled value $w_i = (k_i, p_i)$
2. For each gate G_i with output wire W_c and input wire W_a and W_b we compute the garbled output:

[3]The notion of computation complexity as a performance indicator can be found in Section 4.2

a) If G_i is an XOR:
We set $w_c = (k_c, p_c) = (k_a \oplus k_b, p_a \oplus p_b)$.

b) If G_i is not an XOR:
We decrypt the wire value w_c from the garbled table value at position (p_a, p_b) by means of the random Oracle H:

$$w_c = (k_c, p_c) = H(k_a \| k_b \| i) \oplus e_{p_a, p_b}.$$

3. For each circuit output wire W_i with garbled value $w_i = (k_i, p_i)$ we decrypt the circuit output v_i from the output table value at position p_i by means of the random Oracle H:

$$v_i = H(k_i \| \text{“}out\text{”} \| i) \oplus e_{p_i}.$$

3.3 Linear Secret Sharing

We assume that all secrets will be linearly shared modulo some prime p. Let $x \in \{0,1\}^\ell$ be some (ℓ-bit) secret belonging to some dealer. Then linearly sharing x over n parties $P_1, \ldots, P_n$ means that the dealer picks $n-1$ uniformly random numbers $[x]_i \in \mathbb{Z}_p$, sets $[x]_n = x - \sum_{i=1}^{n-1} [x]_i \pmod p$ and sends share $[x]_i$ to P_i for all $i \in \{1, \ldots, n\}$ over a secure channel. We should note that all shares (separately) are now uniformly random numbers in $\mathbb{Z}_p$. Throughout this paper we will write $[x]$ for a general sharing of some secret x over parties P_i, $i \in \{1, \ldots, n\}$.

Definition 3.3.1. We say that the parties open a value x from sharing $[x]$ if they follow the following steps:

1. Every party i sends their share $[x]_i$ to all other parties $(P_j)_{j \neq i}$.

2. Every party locally computes $x = \sum_{i=1}^{n} [x]_i$.

This definition of *opening* a secret requires that all parties are honest. This scenario makes sense in the semi-honest model described in Section 4.1.

When working in a so called malicious model, where possibly not every party is honest, all shares should come with a special verification tag. Which means that every party can verify every share and calculate the value of x locally. This set-up ensures that an honest party can immediately identify malicious parties by discovering false shares. For this paper we will however focus on the semi-honest model in the two-party setting.

Public value addition
Adding a public value z to a sharing $[x]$ can be done by setting P_1's share $[z+x]_1 = z + [x]_1$ locally and P_i's share remains the same, $[z+x]_i = [x]_i$, for $i \in \{2, \ldots, n\}$.

The correctness follows from opening $[z+x]$:

$$\sum_{i=1}^{n}[z+x]_i = [z+x]_1 + \sum_{i=2}^{n}[z+x]_i = z + [x]_1 + \sum_{i=2}^{n}[x]_i = z + \sum_{i=1}^{n}[x]_i = z + x.$$

Secure addition

The definition of addition $[x+y]$ for given sharings $[x]$ and $[y]$ is rather straightforward. Let $[x]$ and $[y]$ be given over n parties P_n. Then $[x+y]$ can be calculated by letting each party $P_i,\ i \in \{1,\ldots,n\}$ compute $[x+y]_i = [x]_i + [y]_i$. The correctness follows from opening $[x+y]$ when it's calculated in this way:

$$\sum_{i=1}^{n}[x+y]_i = \sum_{i=1}^{n}[x]_i + [y]_i = \sum_{i=1}^{n}[x]_i + \sum_{i=1}^{n}[y]_i = x + y.$$

Public scalar multiplication

Multiplying a public scalar c with a sharing $[x]$ can be done by computing $[cx]_i = c\cdot[x]_i$ locally for every Party P_i, $i \in \{1,\ldots,n\}$. The correctness follows from opening $[cx]$:

$$\sum_{i=1}^{n}[cx]_i = \sum_{i=1}^{n} c[x]_i = c\sum_{i=1}^{n}[x]_i = cx.$$

Secure multiplication

A secure multiplication for linear secret sharing can be done by means of a precomputed shared triplet $([a],[b],[c])$ for which $a \cdot b = c \bmod p$.

1. Each party computes its private share $[e] = [x-a] = [x] - [a]$.
2. Each party computes its private share $[d] = [y-b] = [y] - [b]$.
3. d and e are opened.
4. Since $xy = (a+e)(b+d) = c + ad + eb + de$, each party can (locally) compute its private share by $[xy] = [c] + [a]d + e[b] + de$.

Chapter 4

Security & Performance

In the first section of this chapter we will discuss the meaning of the word *security*. But security is of course not the only desired feature. So in order to evaluate the performance on multiple variants of such secure comparison protocols, we introduce the desired key performance indicators (KPI) in the second section of this chapter.

4.1 Semi-Honest Model

Consider a world without eavesdroppers, we would not have to worry about man-in-the-middle attacks like duplication of messages, alteration of messages or general information leakage due to eavesdropping. This is our primary assumption for the semi-honest model. Furthermore, we consider a party who is participating in a protocol to be semi-honest if it follows all the protocol rules. So suppose that the protocol dictates that Bob should send his value of x, he will not be sending a different message y instead. So what is there left to secure, we might wonder now. Well, what happens if participating parties happen to have to much time on their hands; let's say an infinite amount? Then they start calculating with the values which they might have received/viewed while performing the protocol. Their goal then might be to uncover other peoples secrets from these views. In order to prevent this, or to make sure that this takes them a excruciation long time, we introduce the notion of security in the two party semi-honest model via Definition 4.1.3.

4.1.1 Definition of Privacy

The concepts of security and privacy regarded in this section come from [13].

Definition 4.1.1. We call a function $f : \mathbb{N} \to \mathbb{R}$ *negligible in* n if for every positive polynomial p there exists a $n_0 \in \mathbb{N}$ such that

$$f(n) \leq \frac{1}{p(n)}, \quad \forall n \geq n_0.$$

Definition 4.1.2. (indistinguishability)
Let $X = \{X_i\}_{i \in I}$ and $Y = \{Y_i\}_{i \in I}$ be two families of stochastic variables with (countable) index set I for which every X_n and Y_n ranges (at most) over $\{0,1\}^{\ell(n)} =: V_n$ where $\ell(n) \in \mathcal{O}(n^k)$ for some $k \in \mathbb{N}$. Then we distinguish between three types of indistinguishability:

1. **Perfectly indistinguishable:**
   We call X and Y *perfectly indistinguishable* ($X \overset{p}{\equiv} Y$) when for all $n \in I$:

$$\sum_{\alpha \in V_n} \left| \mathbb{P}(X_n = \alpha) - \mathbb{P}(Y_n = \alpha) \right| = 0.$$

2. **Statistically indistinguishable:**
   We call X and Y *statistically indistinguishable* ($X \overset{s}{\equiv} Y$) when for all $n \in I$:

$$\sum_{\alpha \in V_n} \left| \mathbb{P}(X_n = \alpha) - \mathbb{P}(Y_n = \alpha) \right| \text{ is negligible in } n.$$

3. **Computationally indistinguishable:**
   We call X and Y *computationally indistinguishable* ($X \overset{c}{\equiv} Y$) when for all $n \in I$:

$$\left| \mathbb{P}\left(D_n(X_n) = 1\right) - \mathbb{P}\left(D_n(Y_n) = 1\right) \right| \text{ is negligible in } n,$$

   for every PPT algorithm $D_n : V_n \to \{0,1\}$ (binary distinguisher).

Definition 4.1.3. (Security in the two party semi-honest model)
Let $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$ be a computable function. Notation: $f(x,y) = (f_1(x,y), f_2(x,y))$. Let Π be a two-party protocol for computing f. Then $VIEW_1^\Pi(x,y) = (x, r_1, m_1)$ and $VIEW_2^\Pi(x,y) = (y, r_2, m_2)$ denote the view of party 1 (respectively party 2) during execution of Π under (x,y) where r_i represents the vector of intern coin tosses made by party i and m_i represents the vector of all messages party i received during the protocol. Also $OUTPUT^\Pi(x,y) = \big(OUTPUT_1^\Pi(x,y), OUTPUT_2^\Pi(x,y)\big)$ denotes the output of both parties generated while executing Π under (x,y).

We say that Π computes f securely if there exist probabilistic polynomial-time algorithms S_1 and S_2 (simulators) such that

$$\{S_1\left(x, f_1(x,y)\right), f(x,y)\}_{x,y \in \{0,1\}^*} \equiv \left\{VIEW_1^\Pi(x,y), OUTPUT^\Pi(x,y)\right\}_{x,y \in \{0,1\}^*}$$

$$\{S_2\left(y, f_2(x,y)\right), f(x,y)\}_{x,y \in \{0,1\}^*} \equiv \left\{VIEW_2^\Pi(x,y), OUTPUT^\Pi(x,y)\right\}_{x,y \in \{0,1\}^*}$$

f is said to computed perfectly, statistically or computationally secure when the equivalence relation is given by $\overset{p}{\equiv}$, $\overset{s}{\equiv}$ or $\overset{c}{\equiv}$ respectively. Note that $OUTPUT_i^\Pi(x,y)$ is fully determined by $VIEW_i^\Pi(x,y)$. Also, if f is deterministic we can omit $f(x,y)$ and $OUTPUT^\Pi(x,y)$ from the definition; Π being a protocol which computes f suffices for deterministic f.

Theorem 4.1.1. *Multiplication of linearly shared secrets as described in Section 3.3 is perfectly secure.*

Proof. We show how to construct a general simulator $S_i([a]_i, [b]_i, [c]_i, [x]_i, [y]_i, [xy]_i)$ for $i \in \{1,2\}$ since the protocol is symmetric. Note that at that start of the protocol every share which is unknown to P_i lies uniformly random in $\mathbb{Z}_p$.

1. S_i calculates $[e]_i = [x]_i - [a]_i$ and picks $[e]' \in_R \mathbb{Z}_p$ uniformly at random and sets $e' = [e]_i + [e]'$.

2. If $e' + [a]_i = 0 \bmod p$ then S_i picks $[d]' \in_R \mathbb{Z}_p$ uniformly at random. Otherwise, S_i sets $[d]' = ([xy]_i - [c]_i - [a]_i[d]_i - e'[b]_i - [d]_i e')\,([a]_i + e')^{-1}$

3. S_i outputs its input as well as $[e]'$ and $[d]'$.

Both parties can use the same general simulator because the protocol corresponding to a fixed party is symmetrically equivalent to it's counterpart. The perfect indistinguishability between the simulators and the views follows from the observation that the only truly interesting values $[e]'$ and $[d]'$ are now identically distributed (with respect to the input) as in the instances of views $[e]_{i\oplus 1}$ and $[d]_{i\oplus 1}$. Also the output of the protocol $OUTPUT^{\Pi}(x, y) = xy = f(x, y)$ by the correctness proof already given in Section 3.3. □

The main idea of the semi-honest security definition stated above is that both parties (Alice and Bob) do not deviate from the protocol. They can however perform calculations with their respective views gained from preforming the protocol. The aim for security in this sense is that both parties do not learn to much new information due to these views from the protocol Π. Our goal is now to define a comparison protocol which on homomorphic encrypted inputs $[\![x]\!], [\![y]\!]$ outputs the homomorphic encrypted boolean comparison result $[\![(x < y)]\!]$, where

$$(x < y) := \begin{cases} 1 & \text{if } x < y, \\ 0 & \text{if } x \geq y. \end{cases}$$

Which is secure in the semi-honest sense. It turns out that there exist many such protocols. Therefore we would like to distinguish between *good* and *bad* protocols.

4.2 Key Performance Indicators

Now that we know when a comparison protocol is secure in the Semi honest model, we should discuss when such protocols are considered "good". This will not be done by defining a single attribute to be good, rather we pick key performance indicators (KPI) so we get a transparent view of the performance different protocols might display in different settings.

Communication Complexity

The total communication needed by a protocol can be measured by the number of Kilo Bytes (KB) of data which have to be send over a communication line during a protocol. Obviously, a protocol which enforces 1 GB = 1024^2 KB of communication is not considered a good protocol when compared to another protocol which only needs 100 KB of communication data. Usually, the communication performance of a protocol depends greatly on the bit-length of the input. So in order to keep the performance evaluation fair, one should make sure to optimize the total communication as good as possible considering a given input length ℓ for the protocol.

Round Complexity

In many applications, the number of communication rounds tends to be the bottle neck of a protocol. This is due to the fact that a lot of overhead capacity might be

needed to initiate and terminate a (secure) communication line between the other party. In general, however, this really depends on the communication technology used in practise. We use Toft's definition of a *round* of communication [29], because it gives us a workable and intuitive definition of this concept.

Toft argues that a communication round consists of sending information to other parties and performing a limitless number of arithmetic computations with a sole restriction: variables which are received by a party during this round can not be used in any arithmetic operation performed by that party in the same round.

Example 1. Let Alice hold a secret value $s_A \in \mathbb{R}$, let Bob also hold a secret value $s_B \in \mathbb{R}$ and we assume to have some public parameter $p \in \mathbb{R}$. Then the following protocol takes only one round.

cross-message example

1. Alice computes $x = p \cdot s_A$ en sends it to Bob.
2. Bob computes $y = p \cdot s_B$ en sends it to Alice.

Such a simple cross-message protocol costs only one round because neither party used any information from their received messages x, y during their computation. Now consider the following standard question & reply protocol.

question & reply example

1. Alice computes $x = p \cdot s_A$ en sends it to Bob.
2. Bob computes $y = s_B \cdot x$ en sends it to Alice.

This protocol will take two rounds because we have to close the previous round and open a new one before Bob may use the received value x in his computation of y.

Example 2. The secure multiplication protocol from Section 3.3 takes one communication round. Opening d and e is a classic example of cross-message sending. Although the actual calculation needed to open d and e and computing $[xy]$ can not be performed during the same round, we note that no more actual communication is needed. Hence we can postpone these calculations until we open a new communication round needed by another protocol.

Bandwidth

The bandwidth of a protocol is given by the maximum number of bits send during a single communication round. This performance indicator strikes an interesting issue consider the previous two KPI. When the communication complexity of a protocol doesn't change, but we manage to decrease the round complexity, then odds are

that the necessary bandwidth for the protocol will increase. This is obviously not always the case, but shows that we might encounter some trade off considering these KPI. Note that when one finds the necessary bandwidth of a protocol to hight in practise, one can always implement the round(s) with the highest communication complexity as two separate communications to decrease the bandwidth.

Computational Complexity

This KPI is mentioned way less in recent literature than the previous ones. Which is one of the main reasons why this research project exists in the first place. The Computational complexity is measured as the amount of time it takes ones CPU to compute the desired result of a protocol. This KPI is shunned so often as it takes a lot of extra time and effort to implement the proposed protocol. To our knowledge, the only secure comparison protocol, described in the next chapter, for which the computation complexity was studied previous to our research, was that of Garbled Circuits [19].

Chapter 5

Secure Comparison Protocols in the Two-Party Semi-Honest Model

In this chapter we will discuss multiple secure-comparison protocols. In order to ensure a fair comparison between the different protocols we enforce the following general Paillier encryption setting:

Party	Alice	Bob
Public	pk	
Private	sk	$[\![x]\!]$ and $[\![y]\!]$
Output	$-$	$[\![(x < y)]\!]$

We consider three main steps during our comparison protocols:

1. The encrypted inputs $[\![x]\!]$,$[\![y]\!]$ belonging to Bob are converted to two inputs $c, r < 2^\ell$ held privately by Alice and Bob respectively. This conversion is described below.

2. Alice and Bob securely (bit-)compare r and c according to the given setting with result $\epsilon = (c < r)$. In what format ε is computed depends on the given setting.

3. Alice and Bob convert their private information back to the comparison result $[\![\delta]\!] = [\![(x < y)]\!]$.

Step 1. is achieved by applying the following protocol:

Input Conversion

1. Bob picks a random mask $\rho \in 1||\{0,1\}^{\ell+\kappa-1}$ for $z := x + 2^\ell - y$. He then sends $[\![\gamma]\!] = [\![\rho + z]\!] = [\![\rho + 2^\ell + x - y]\!] = [\![\rho + 2^\ell]\!][\![x]\!][\![y]\!]^{-1}$ to Alice.

2. Alice decrypts γ en computes $c = (\gamma \bmod 2^\ell)$.

3. Bob computes $r = (\rho \bmod 2^\ell)$.

This first step will be the same in all protocols. It requires that $2^{\kappa+\ell+3} < n$ in order to prevent overflow, where κ is the statistical security parameter[1], ℓ the size of the input and n the modulus of the Cypher space. This requirement is met because in order for the homomorphic encryption to be sufficiently secure an large RSA modulus is provided. Step 2. and 3. will greatly depend on the specific settings and will therefore be discussed later.

5.1 Homomorphic Encryption

Because $\delta = 1 - z_\ell$ we seek a method to calculate z_ℓ. This can be done by defining the integer division $d : \mathbb{N} \to \mathbb{N}$ given by $d(n) = \lfloor n/2^\ell \rfloor$, together with a secure way of computing such an integer division. In [30] a secure integer division by blinding is defined. Note that we have already done the first part of this protocol, namely the blinding of z with ρ with result $\gamma = \rho + z$. Also the modulo reductions of γ and ρ by 2^ℓ (r and c) are already known to Alice and Bob respectively. So the only thing we need to do, in order perform the integer division of z by 2^ℓ, is to compute

$$z_\ell = d(z) = d(\gamma) - d(\rho) - (c < r). \tag{5.1}$$

Proof. We know

$$d(\gamma) \cdot 2^\ell + c = \gamma = z + \rho = d(z) \cdot 2^\ell + (z \bmod 2^\ell) + d(\rho) \cdot 2^\ell + r. \tag{5.2}$$

So we see that

$$d(\gamma) = \begin{cases} d(z) + d(\rho) & \text{if } (z \bmod 2^\ell) + r < 2^\ell, \\ d(z) + d(\rho) + 1 & \text{otherwise.} \end{cases}$$

i.e. $d(\gamma) = d(z)+d(\rho)+t$, where t represents the overflow we find from $(z \bmod 2^\ell)+r$ modulo 2^ℓ. Also, from Equation 5.2 it follows that

$$(z \bmod 2^\ell) + r < 2^\ell \iff c = (z \bmod 2^\ell) + r \iff c \geq r.$$

Hence,

$$d(\gamma) = \begin{cases} d(z) + d(\rho) & \text{if } c \geq r, \\ d(z) + d(\rho) + 1 & \text{if } c < r. \end{cases}$$

Which proofs Equation (5.1). □

Remember that we were interested in an *encrypted* comparison result. By combining the fact $\delta = 1 - z_\ell$ and Equation (5.1), we get

$$[\![\delta]\!] = [\![1 - z_\ell]\!] = [\![1 - d(\gamma) + d(\rho) + \varepsilon]\!] = [\![1 + d(\rho)]\!][\![d(\gamma)]\!]^{-1}[\![\varepsilon]\!] \pmod{n^2}.$$

Where $\varepsilon = (c < r)$ is the result of the (bit-)comparison of c and r. Which means we can now safely continue to the second step of our secure comparison protocol.

For this homomorphic variant we compute $[\![\varepsilon]\!]$. By applying a dedicated DGK protocol presented in [16]. So we assume the following DGK ($\mathcal{D}$) and Paillier ($\mathcal{P}$) encryption setting:

[1] $\kappa = 40$ is sufficient in practise

Party	Alice	Bob
Public	$pk_{\mathcal{P}}$, $pk_{\mathcal{D}}$	
Private	$sk_{\mathcal{P}}$, $sk_{\mathcal{D}}$, c	r
Output	–	$[\![(c < r)]\!]_{\mathcal{P}}$

Where both c and r have at most length ℓ. Then the following steps compute and supply Bob with the result $[\![\varepsilon]\!]_{\mathcal{P}} = [\![(c < r)]\!]_{\mathcal{P}}$ in a secure way.

HE comparison

1. Alice encrypts the bits of $c = (c_{\ell-1}, \ldots, c_0)_2$ as $[\![c_{\ell-1}]\!]_{\mathcal{D}}, \ldots, [\![c_0]\!]_{\mathcal{D}}$ and sends them to Bob.

2. Bob picks a secret $s \in_R \{-1, 1\}$ uniformly at random and computes the following values:

$$[\![e_i]\!]_{\mathcal{D}} = [\![s + r_i - c_i + 3 \sum_{j=i+1}^{\ell-1} c_j \oplus r_j]\!]_{\mathcal{D}}\,, \quad \text{and} \quad [\![e_\ell]\!]_{\mathcal{D}} = [\![s - 1 + 3 \sum_{j=0}^{\ell-1} c_j \oplus r_j]\!]_{\mathcal{D}}$$

   for $0 \leq i < \ell$ and $r = (r_{\ell-1}, \ldots, r_0)_2$.

3. Bob re-randomizes the messages and cyphers corresponding to the e's without destroying zero messages, to prevent Alice of reverse engineering relevant information. This is done by picking $\ell + 1$ dummy-messages $m_0, \ldots m_\ell \in \mathbb{N}_{<u} \setminus \{0\}$ and randomizers $w_0, \ldots, w_\ell \in \{0, 1\}^{2t}$ and setting

$$[\![e'_i]\!]_{\mathcal{D}} = [\![e_i]\!]_{\mathcal{D}}^{m_i} \cdot h^{w_i},$$

   for $0 \leq i \leq \ell$.

4. Bob then sends the $[\![e'_i]\!]_{\mathcal{D}}$'s in a random permuted order to Alice.

5. Alice checks whether there are zero's amongst the e'_i's by checking whether

$$[\![e'_i]\!]_{\mathcal{D}}^{v} \equiv 1 \pmod{n}$$

   (see Lemma 3.1.4). Alice sends $[\![\tau]\!]_{\mathcal{P}}$ to Bob for which

$$\tau = \begin{cases} 0 & \text{if there is a zero message amongst the } e'_i\text{'s,} \\ 1 & \text{otherwise.} \end{cases}$$

6. Bob corrects the encrypted non-zero query bit τ according to the value of his secret s:

$$[\![\varepsilon]\!] = [\![s \oplus \tau]\!] = \begin{cases} [\![\tau]\!] & \text{if } s = 1, \\ [\![1 - \tau]\!] = [\![1]\!][\![\tau]\!]^{-1} & \text{if } s = -1. \end{cases}$$

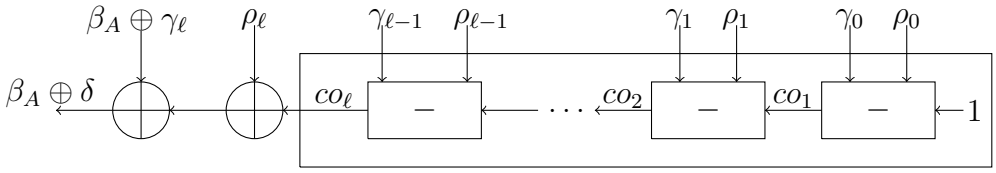

Figure 5.1: Subtraction circuit derived from [17].

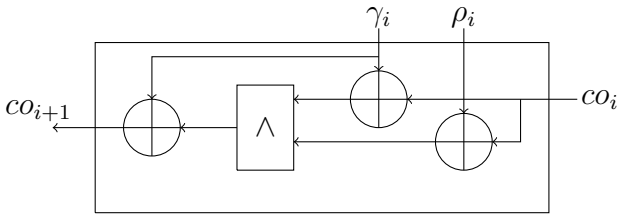

Figure 5.2: 1-bit subtraction circuit from [17].

The random permutation, mentioned in step 4, can be efficiently implemented by a Fisher-Yates shuffle.

5.2 Garbled Circuits

In this section we discuss the design of the bit-comparison circuit using the construction from [17] as our base. They describe a circuit which is very similar to their subtraction circuit to compute the comparison bit $\delta = 1 - z_\ell$. In essence they compute compute $z = \gamma - \rho$ up until the $(\ell+1)$'th bit which is similar to computing $\varepsilon = (c < r)$, as we did in the previous section. However, our setting is a bit different. We don't want to disclose the value of δ to either party. So we propose an efficient alteration to the original circuit to convert the output of the protocol to the Pailler encrypted $[\![\delta]\!]$. To achieve this, we let Alice generate a random bit β_A, which she presents to the system by setting her last input wire value as $\beta_A \oplus \gamma_\ell$ instead of γ_ℓ. Also, she sends a Paillier encrypted $[\![\beta_A]\!]$ to Bob in a communication round of the $\binom{2}{1}$-OT pre-computation. This way, Bob will end up with $\beta_A \oplus \delta$ after evaluation of the circuit. Which will tell him whether to invert the Paillier encrypted coin toss or not:

$$[\![\delta]\!] = \begin{cases} [\![\beta_A]\!] & \text{if } \beta_A \oplus \delta = 0, \\ [\![1 - \beta_A]\!] = [\![1]\!][\![\beta_A]\!]^{-1} & \text{if } \beta_A \oplus \delta = 1. \end{cases}$$

The resulting circuit is depicted in Figure 5.1 together with Figure 5.2.

5.3 Secret Sharing

Every secure multiplication $f([x], [y]) = [xy]$ performed by two parties (Alice and Bob) needs a shared multiplication triplet $([a], [b], [c])$ for which we have $c = a \cdot b \bmod p$. Using such a triple for more than one multiplication is insecure. So in general Alice and Bob might need a lot of these triples in order to perform a secure comparison protocol. But we are in luck, for we can precomputed such triplets because of their independence to x and y.

Generating secure multiplication triplets in the Semi-honest two-party setting

This is done by the following protocol:

1. Alice picks $[a]_1, [b]_1 \in_R \mathbb{Z}_p$ uniformly at random, encrypts them according to some additive homomorphic encryption and sends the encryptions $[\![[a]_1]\!]$, $[\![[b]_1]\!]$ to Bob.
2. Bob picks $[a]_2, [b]_2 \in_R \mathbb{Z}_p$ and a $r \in \{0,1\}^{2\ell(p)+\kappa+1}$ uniformly at random, where κ ($= 40$ in practise) is the statistical security parameter.
3. Bob calculates $[\![z]\!] = [\![[a]_1 \cdot [b]_2 + [b]_1 \cdot [a]_2 + r]\!] = [\![[a]_1]\!]^{[b]_2} \cdot [\![[b]_1]\!]^{[a]_2} \cdot [\![r]\!]$ and sends the result to Alice.
4. Alice decrypts and computes $[c]_1 = [a]_1[b]_1 + z \bmod p$.
5. Bob sets $[c]_2 = [a]_2[b]_2 - r \bmod p$.

Correctness follows from the following observation:

$$c = [c]_1 + [c]_2 = [a]_1[b]_1 + z + [a]_2[b]_2 - r \pmod p$$

and

$$z - r = [a]_1 \cdot [b]_2 + [b]_1 \cdot [a]_2 \pmod p.$$

Hence

$$c = ([a]_1 + [a]_2)([b]_1 + [b]_2) = a \cdot b \pmod p.$$

Note that using a Paillier scheme would be relatively inefficient because of its large message space. Therefore we decided to implement the pre-computation of multiplication triplets with the Okamoto-Uchiyama scheme, which has a smaller message space, making decryption faster.

In the following sections we will discuss three different protocols which perform the bitwise comparison $\varepsilon = (c < r)$ and store the result in two shares $[\varepsilon]_A$, $[\varepsilon]_B$ which will belong to Alice and Bob respectively. The transformation to the comparison result $[\![\delta]\!]$ is then achieved via the following protocol:

Transforming result from Secret Sharing to Homomorphic Encryption

1. Alice encrypts her share: $[\![[\varepsilon]_A]\!]$ and sends it to Bob.
2. Bob computes

$$[\![\varepsilon]\!] = \begin{cases} [\![[\varepsilon]_A]\!] & \text{if } [\varepsilon]_B = 0, \\ [\![1 - (p-1)^{-1}[\varepsilon]_A]\!] = [\![1]\!] \cdot [\![[\varepsilon]_A]\!]^{-(p-1)^{-1}} & \text{if } [\varepsilon]_B = 1, \\ [\![[\varepsilon]_A + [\varepsilon]_B - p]\!] = [\![[\varepsilon]_A]\!] \cdot [\![[\varepsilon]_B]\!] \cdot [\![p]\!]^{-1} & \text{if } [\varepsilon]_B > 1, \end{cases}$$

where $(p-1)^{-1}$ is the inverse of $(p-1) \in \mathbb{Z}_n^*$.

3. Alice computes $d(\gamma) = \lfloor \gamma / 2^\ell \rfloor$ and sends it to Bob.
4. Bob computes $[\![\delta]\!] = [\![1 - z_\ell]\!]$ via Equation (5.1).

In order to see that step 2, in the protocol above, has the desired result, we note that $[\varepsilon]_A + [\varepsilon]_B \in \{0, 1, p, p-1\}$ and $p < n$.

In the following linear secret sharing settings we would require c to be public rather than a privately know to Alice. So we propose the following transformation introduced by Nishide an Ohta [22] which also reduces the size of the needed bitwise comparison:

Preparation GSV07/NO07

1. Alice and Bob generate $\ell(p) = \lceil \log_2(p) \rceil$ shared random bits $[u_i]$ such that $u = (u_{\ell(p)-1}, \ldots, u_0)_2 < p$.
2. Alice sets $[v]_A = c$, and Bob sets $[v]_B = (-r \bmod p)$ which results in the sharing $[v] = [c - r]$.
3. They set $[d] = [2v + u] = 2[v] + [u]$ and open it.

Lemma 5.3.1. *If $c, r < p/2$, then*

$$\varepsilon = 1 - \left(v < \frac{p}{2} \right).$$

Proof. the proof of this Lemma is very straightforward and follows immediately from the assumption $c, r < p/2$ when we observe

$$v = c - r + (c < r)p.$$

□

With Lemma 5.3.1 we can proof something stronger which allows us to securely compute ε by means of only one secure multiplication.

Theorem 5.3.1. *If $c, r < p/2$, then*

$$\varepsilon = d_0 \oplus u_0 + (d < u) - (d_0 \oplus u_0) \cdot (d < u),$$

where d_0 and u_0 are the least significant bits of d and u respectively.

Proof. By Lemma 5.3.1 we have

$$\varepsilon = 1 - \left(v < \frac{p}{2}\right),$$

where the right hand side equals the least significant bit of $2v$, because p is prime. Now we observe that

$$d_0 = \begin{cases} (2v)_0 \oplus u_0 & \text{if } d \geq u, \\ 1 \oplus (2v)_0 \oplus u_0 & \text{if } d < u. \end{cases}$$

By the construction of d. Hence, we can rewrite:

$$(2v)_0 = (d < u) \oplus d_0 \oplus u_0 = d_0 \oplus u_0 + (d < u) - (d_0 \oplus u_0) \cdot (d < u).$$

Which proofs the theorem. □

Note that picking $p > 2 \cdot (2^\ell - 1)$, where ℓ is the length of the inputs x and y is sufficient for the above theorem to hold. So for linear secret sharing we will, from now on, only consider $p > 2^{\ell+1} - 2$. This way the only remaining problem is to find $(d < u)$ from the bitwise shared u and public d. This threshold for p is achieved in the implementation, without increasing the data complexity to unnecessary high values, by setting p such that

$$p = \min_k \left\{ \max_q \left\{ q \in \{0,1\}^k \ : \ q \text{ is prime, and } q > 2^{\ell+1} - 2 \right\} \right\}.$$

Now that we have established the appropriate size of our prime p we need to solve another small issue concerning the preparation protocol. In step one Alice and Bob need the generate a bitwise shared random value. They do this by successively generating random shared bits $[u_i]$ by means of the protocol below. Then they set

$$[u] = \sum_{i=0}^{\ell-1} [u_i] \cdot 2^i.$$

After which they perform a bitwise comparison to check whether $u < p$. If the underlying secure comparison protocol claims a constant number of rounds (like NO07), one should make sure that enough u's are generated in advance such that the protocol will have negligible abort-probability. Where we assume that the protocol will abort when no appropriate u is found. The subject on abort-probability is therefore further explained in Subsection 5.3.2.

Generating a shared random bit

1. Alice and Bob both perform a random coin toss with fair coins resulting in random bits b_A and b_B.
2. They both create a sharing of their bit by setting the other's share to zero, i.e.
$$[b_A] = (b_A, 0), \quad [b_B] = (0, b_B).$$

3. They compute the random shared bit

$$[b] = [b_A \oplus b_B] = [b_A] + [b_B] - 2[b_A] \cdot [b_B],$$

by means of secure multiplication.

The reader should be aware that this protocol relies heavily upon the fact that we work in the semi honest model and therefore needs to be replaced with something more elaborate, such as found in [22], when the security settings are more complicated.

In the following subsections we will discuss three different approaches for a bitwise comparison. The first approach, by Garay, Schoenmaker and Villega [12] will use a linear number of communication rounds and grants perfect security by full blinding. The second approach by Nishide and Ohta [22] will also grant perfect security and uses a constant number of communication rounds. The third approach by Catrina and de Hoogh [7] enforces a larger prime p which results in statistical security and a constant number of communication rounds.

5.3.1 GSV07

The first bitwise comparison we consider by Garay, Schoenmaker and Villega [12] uses a relatively small prime $p > 2^{\ell+1} - 2$. They compute the comparison bit $\lambda_{\ell(p)} = (d < u)$ by means of the following recurrence relation:

$$\lambda_0 = 0, \qquad \lambda_{i+1} = (1 - (u_i - d_i)^2)\lambda_i + u_i(1 - d_i).$$

In our setting, $d = (d_{\ell(p)-1}, \ldots, d_0)_2$ is publicly known and u is bitwise shared between Alice and Bob. So the GSV07 bitwise comparison protocol is then performed by initiating the shared value

$$[\lambda] \leftarrow [u_0(1 - d_0)] = \begin{cases} [0] & \text{if } d_0 = 1, \\ [u_0] & \text{if } d_0 = 0, \end{cases}$$

and refilling the value for $i = 1, \ldots, \ell(p) - 1$ in the following way:

$$[\lambda] \leftarrow \begin{cases} [u_i][\lambda] & \text{if } d_i = 1, \\ (1 - [u_i])[\lambda] + [u_i] & \text{if } d_i = 0. \end{cases}$$

5.3.2 NO07

The bitwise comparison protocol as proposed by Nishide and Ohta also uses a relatively small prime $p > 2^{\ell+1} - 2$. The main difference with the previous protocol is that it runs in a fixed number of communication rounds as opposed to the linear number of communication rounds needed in the GSV07 protocol.

The NO07 protocol is made up out of sub-protocols, namely "Unbounded Fan-In Or" and "Prefix-Or".

Unbounded Fan-In Or

The length k Unbounded Fan-In Or $[\bigvee_{i=0}^{k-1} a_i]$ is calculated from input $([a_i])_{i=0}^{k-1}$ as follows:

1. Determine the unique k-degree polynomial $f_k \in \mathbb{Z}_p[X]$, for which $f_k(1) = 0$ and $f_k(x) = 1$ for all $x \in \{2, 3, \ldots, k+1\}$, by Lagrange Interpolation.
2. Compute $[A] = 1 + \sum_{i=0}^{k-1} [a_i]$.
3. Generate $[b_i \in \mathbb{Z}_p]$ and $[b'_i \in \mathbb{Z}_p]$ for all $i \in \{0, \ldots, k-1\}$.
4. Compute $[B_i] = [b_i] \cdot [b'_i]$ in parallel.
5. Open the B_i's and compute $[b_i^{-1}] = B_i^{-1}[b'_i]$ in parallel.
6. Compute for $i \in \{0, \ldots, k-1\}$:

$$[c_i] = \begin{cases} [A] \cdot [b_0^{-1}] & \text{if } i = 0, \\ [A] \cdot [b_{i-1}] \cdot [b_i^{-1}] & \text{if } i \geq 1. \end{cases}$$

7. Compute $[A^i] = \left(\prod_{i=0}^{k-1} c_i\right) [b_i]$.
8. Finally we compute the result:

$$\left[\bigvee_{i=0}^{k-1} a_i\right] = [f_k(A)] = \alpha_0 + \sum_{i=1}^{k} \alpha_i [A^i]$$

where $f_k = \sum_{i=0}^{k} \alpha_i X^i$.

The idea of the Unbounded Fan-In Or protocol is that we determine a value A such that no information is leaked when evaluation $f_k(A)$, which would happen if A would attain the value zero. We should also note that the construction of this protocol originates from [10] where they show how to securely compute symmetric boolean functions. Also, constructions from [3] where used which introduces the notion of fault-tolerant computation. Indeed, when we take a closer look at step 5, we see that the protocol might fail due to a non-invertible B_i. When this happens, we let the protocol abort. It is argued that this can be tolerated due to negligible probability of abort. In practise though, when performing many Unbounded Fan-In Or protocols successively, this might not be the case. In order to circumvent this problem we have chosen to implement this with a buffer of extra pairs (b_i, b'_i). We chose that the probability of abort would be considered negligible if it is below $2^{-\kappa}$, where κ is again the statistic security parameter. The needed buffer size is then found by computing the abort-probability distribution by means of the corresponding binomial distribution. In addition, one should not decrease the size of p to much because then the buffer will increase to much. Taking p at least $2^7 - 1$ proofs to be

sufficient for our purpose.

We continue with the Prefix-Or protocol which makes use of the Unbounded Fan-In Or.

Prefix-Or

The length k Prefix-Or $([b_n])_{n=0}^{k-1}$ where $b_n = \bigvee_{i=0}^{n} a_i$ is calculated from input $([a_i])_{i=0}^{k-1}$ as follows:

1. Compute $\lambda = \min\{x \ : \ k \leq x^2\}$.
2. Restructure the input as a $\lambda \times \lambda$ matrix:
$$A = \begin{bmatrix} a_0 & \dots & a_{\lambda-1} \\ a_\lambda & \dots & a_{2\lambda-1} \\ \vdots & & \vdots \\ a_{\lambda(\lambda-1)} & \dots & a_{\lambda^2-1} \end{bmatrix},$$
where $A_{i,j} = a_{i-1+\lambda(j-1)} = 0$ if $i - 1 + \lambda(j-1) > k$.
3. Compute the 'row'-Or $[x_i] = \left[\bigvee_{j=0}^{\lambda} A_{i,j}\right]$ for $i \in \{0, \dots, \lambda - 1\}$ in parallel by using Unbounded Fan-In Or.
4. Compute $[y_n] = [\bigvee_{i=0}^{n} x_i]$ for $n \in \{0, \dots, \lambda - 1\}$ in parallel by using Unbounded Fan-In Or again.
5. Set
$$\begin{bmatrix} [f_0] \\ [f_1] \\ \vdots \\ [f_{\lambda-1}] \end{bmatrix} = \begin{bmatrix} [x_0] \\ [y_1] - [y_0] \\ \vdots \\ [y_{\lambda-1}] - [y_{\lambda-2}] \end{bmatrix}.$$
6. Compute
$$[\alpha_j] = \sum_{i=0}^{\lambda} [f_i] \cdot [A_{i,j}]$$
for $j \in \{0, \dots, \lambda - 1\}$ in parallel.
7. Next, perform another Unbounded Fan-In Or to compute $[\beta_n] = \left[\bigvee_{j=0}^{n} \alpha_j\right]$ in parallel.
8. Finally, set $[s_i] = [y_i] - [f_i]$ for $i \in \{0, \dots, \lambda - 1\}$.
9. Compute the prefix-or output bits as a $\lambda \times \lambda$ matrix:
$$B_{i,j} = [f_i] \cdot [\beta_j] + [s_i].$$
10. Restructure the desired output:
$$b_n = b_{i-1+\lambda(j-1)} = B_{i,j},$$
for $n \in \{0, \dots, k - 1\}$.

Notice that in step 4 we calculate y_n such that $y_n = 1$ if and only if there is a row in A with index $i < n$ for which $x_i = 1$. Then, in step 5, $f_i = 1$ if and only if row i is the first row for which $x_i = 1$, let i_0 be the index of this row. Then step 6 computes exactly that row: $\alpha_j = A_{i_0,j}$. Finally, after computing the fan-in or of the i_0'th row (without revealing i_0), we let $s_i = 1$ if and only if $i > i_0$. Which enables us to compute the output in step 9.

Now that we can preform a Prefix-Or protocol, we can move to the proposed bit-comparison, which is given by the following protocol:

Bitwise Comparison NO07

1. Compute for all $i \in \{0, \ldots, \ell(p) - 1\}$

$$[a_i] = [d_i \oplus u_i] = \begin{cases} [1 - u_i] = [1] \cdot [u_i]^{-1} & \text{if } d_i = 1, \\ [u_i] & \text{if } d_i = 0, \end{cases}$$

   in parallel.

2. Compute $[b_i] = \left[\bigvee_{j=i}^{\ell(p)-1} a_i\right]$ for all $i \in \{0, \ldots, \ell(p) - 1\}$ by Prefix-Or.

3. Set $e_i = [b_i - b_{i+1}]$ for all $i \in \{0, \ldots, \ell(p) - 2\}$ and $e_{\ell(p)-1} = [b_{\ell(p)-1}]$.

4. The result is found by computing

$$[(d < u)] = \sum_{i=0}^{\ell(p)-1} [e_i] \cdot [u_i],$$

   in parallel.

The Lagrange polynomials needed to implement this protocol upto an input length of $\ell = 25$ are given in Appendix A.2.

5.3.3 CH10

In the protocol suggested by Catrina and de Hoogh, p is larger than in the previous approaches, namely $\ell(p) = \kappa + \ell + 3$. This means that we can perform additive blinding without causing carry overs. So we will not be needing the result from Theorem 5.3.1 to find ε. Instead, this protocol will compute the shared comparison bit $[\varepsilon] = [(c < r)]$ a bit more directly.

Preparation CH10

1. Alice and Bob generate ℓ shared random bits $[u_0], \ldots, [u_{\ell-1}]$ and set
$$[u] = [(u_{\ell-1}, \ldots, u_0)_2] = \sum_{i=0}^{\ell-1} [u_i] \cdot 2^i.$$
2. Alice and Bob create a shared random number $\theta < 2^\kappa$, and compute
$$[u'] = [\theta \cdot 2^\ell + u] = [\theta] \cdot 2^\ell + [u].$$
3. Alice sets $[v]_A = c$, and Bob sets $[v]_B = 2^\ell - r$ which results in the sharing $[v] = [2^\ell + c - r]$.
4. They set $[d] = [v + u'] = [v] + [u']$ and open it.

Because there was no carry over in the blinding of v, we can now simply apply Equation 5.1 to find:

$$\varepsilon = (1 - v_\ell) = 1 - d(d) + d(u') + (d \bmod 2^\ell < u' \bmod 2^\ell) = 1 - \left\lfloor \frac{d}{2^\ell} \right\rfloor + \theta + (d \bmod 2^\ell < u).$$

Then the protocol by Catarina and de Hoogh [7], based on observations made by Reistad and Toft [26], computes the bitwise comparison $(d \bmod 2^\ell < u)$ in a constant number of communication rounds. This is done by computing

$$\alpha_i = (u_i \oplus d_i)(1 - d_i)\mu_{i+1} \quad \text{where} \quad \log_2(\mu_i) = \sum_{j=i}^{\ell-1} (u_i \oplus d_i).$$

But before we present this as an actual protocol, we need two more building blocks. Firstly we need Prefix Multiplication.

Prefix Multiplication

The length k prefix multiplication $([p_n])_{n=0}^{k-1}$ where $p_n = \prod_{i=0}^{n} a_i$ is calculated from input $([a_i])_{i=0}^{k-1}$ as follows:

1. Generate $[b_i \in \mathbb{Z}_p]$ and $[b'_i \in \mathbb{Z}_p]$ for $i \in \{0, \ldots, k-1\}$.
2. Compute $[B_i] = [b_i] \cdot [b'_i]$ in parallel.
3. Open the B_i's and compute $[b_i^{-1}] = B_i^{-1}[b'_i]$ in parallel.

4. Compute for $i \in \{0, \ldots, k-1\}$:

$$[c_i] = \begin{cases} [b_0] & \text{if } i = 0, \\ [b_i] \cdot [b_{i-1}^{-1}] & \text{if } i \geq 1. \end{cases}$$

5. Compute $[A_i] = [c_i] \cdot [a_i]$.

6. Open the A_i's.

7. Finally we compute the result:

$$[p_i] = \begin{cases} [a_0] & \text{if } i = 0, \\ [b_i^{-1}] \cdot \prod_{j=0}^{i} A_i & \text{if } i \geq 1. \end{cases}$$

Note that this Prefix Multiplication is fairly similar to the Unbounded Fan-In Or presented in Subsection 5.3.2, due to the fact that the same principles are used when securely computing a symmetric boolean function. So again, we consider the fault-tolerant attribute of the protocol. This time however, we are in bit more luck. Because p is so much larger now, the probability of encountering a non-invertible element B_i becomes extremely small by construction. In order to check whether the abort-probability indeed remains small enough (below 2^{-40}), one should again compute the corresponding binomial distribution. This is now less of a concern in practice however, due to the size of p.

The next building block we will be needing for the CH10 bitwise comparison is a Least Significant Bit (LSB) protocol. As we have large p, we can take the relatively easy protocol as presented in [14].

Least Significant Bit (LSB)

The shared least significant bit $[a_0]$ of $[a] = [(a_{k-1}, \ldots, a_0)_2]$ is retrieved in the following manner:

1. Pick two masks $m_A, m_B \in \{0,1\}^{k+\kappa-1}$, where κ is the statistical security parameter ($\kappa = 40$ in practice), and create a random shared bit $[b]$.

2. Alice computes

$$[t]_A = [a]_A + 2 \cdot m_A + [b]_A,$$

and Bob computes

$$[t]_B = [a]_B + 2 \cdot m_B + [b]_B,$$

to create the sharing $[t]$.

3. Open t and compute

$$[a_0] = t_0 \oplus [b] = t_0 + [b] - 2t_0[b].$$

Note that this LSB protocol does not work in general when confronted with smaller p's due to the risk of overflow.

Now that all the building blocks are present, we end this section with our last Bitwise comparison alternative.

Bitwise Comparison CH10

1. Compute
$$[a_i] = [u_i \oplus d_i] = \begin{cases} [1 - u_i] = [1] \cdot [u_i]^{-1} & \text{if } d_i = 1, \\ [u_i] & \text{if } d_i = 0, \end{cases}$$
in parallel.
2. Compute $[\mu_i] = \prod_{j=i}^{\ell-1}(a_i + 1)$ via prefix multiplication.
3. Set $e_i = [\mu_i - \mu_{i+1}]$ for all $i \in \{0, \ldots, \ell(p) - 2\}$ and $e_{\ell-1} = [\mu_{\ell-1}] - 1$.
4. Set $[\alpha_i] = [(1 - d_i)e_i] = \begin{cases} [0] & \text{if } d_i = 1, \\ [e_i] & \text{if } d_i = 0, \end{cases}$
5. Compute
$$[E] = \sum_{i=0}^{\ell-1} [\alpha_i]$$
6. The result is then given by running the LSB protocol on E with output $E_0 = (d \bmod 2^\ell < u)$.

We observe that the a_i, as computed in step 1, equals μ_{i+1} if and only if $d_i < u_i$. In addition, $\mu_{i+1} = 1$ if and only if the $\ell - i$ leftmost (most significant) bits of $d \bmod 2^\ell$ and u are equal. So it follows that there exists exactly one i_0 for which $a_{i_0} = 1$ if and only if $d \bmod 2^\ell < u$. Because in all other cases, either $\alpha_i = 0$ or μ_{i+1} is even. Hence the computed E from step 5 is odd if and only $d \bmod 2^\ell < u$.

Chapter 6

Results

In this chapter we present the experimental results of the secure comparison protocols from Chapter 5 measured in the key performance indicators as defined in Section 4.2.

The code of the implementation was written in C/C++ with the use of several external libraries, namely MPIR[1], Boost[2], OpenSSL[3] and SeComLib[4]. MPIR stands for *Multiple Precision Integers and Rationals* and is used to enable arithmetic operations on integers of large bit-length. This software is needed because C/C++ does not, in general, support integer values of more than 32 bits. The OpenSSL library includes implementation of Hash functions which is needed for Garbled Circuits. SeComLib is an open source library, made by Mihai Todor, from the Cyber Security Group at Delft University of Technology. This library includes built-in functionality on homomorphic encryption schemes as discussed in Section 3.1. Boost is included as a dependency of SeComLib.

The cryptographic key lengths where chose such that they meet the current standard set by the American National Institute of Standards and Technology (NIST)[4], valid up till 2030, as given in Table 6.1. The statistical security parameter κ was set to 40 bits. The Paillier keys were taken to be same for every implementation, to provide a fair comparison. The value of these keys can be found in Appendix A.3. The experiments were conducted on a machine running Windows 7 Enterprise, 64-bit operating system, with an Intel(R) Core(TM) i3 CPU M 330 @ 2.13GHz and 4 GB of RAM. The input size ℓ was varied from 1 to 25 bit. Each setting was initiated (computations which do not depend on the input size) once for ever input size and then run 100 successive times. Each of the 100 runs was split into its pre-computation phase (computations for which no knowledge of the actual input is necessary) and its on-line phase (computations which must be run after receiving the actual input).

The number of rounds which is required for each protocol is given in Table 6.2. It shows that all but one protocols have constant number of rounds. Only GSV07 has a linear round complexity relative to the bit-size of the input.

[1] `http://www.mpir.org`
[2] `http://www.boost.org`
[3] `https://www.openssl.org`
[4] `http://cybersecurity.tudelft.nl/content/secomlib`

Table 6.1: Key Lengths

Security strength	Asymmetric key	Discrete Log key	Discrete Log Group	Hash
112	2048	224	2048	SHA-256

Table 6.2: Number of rounds in each protocol.

	H	**GC**	**GSV07**	**NO07**	**CH10**
Initialization	1	1	1	1	1
Pre-processing	-	2	$3 + \log_2(p)$	14	4
Online computation	4	2	$3 + \log_2(p)$	13	6

6.1 Initialization

The Initialization phase consists of calculations which are not depended on the input length and have to be performed only once in order to run the protocol multiple times. This includes computations like key generating, variable initiation and memory locking. Table 6.3 shows the average computation complexity for this stage for all protocols.

Table 6.3: Average time required for initialisation.

	GC	**H**	**GSV07**	**NO07**	**CH10**
Time (s)	0.0071	144	4.1	4.1	4.1

As mentioned in Section 3.2.2 The time for generating a Sophie Germain prime q (in GC) is not counted here but fixed to the value shown in Appendix A.1. The search for a generator g of G_q is however included.

6.2 Pre-Computation

The pre-computation phase consists of all computations which may depend on the input length ℓ, but do not depend on the input values themselves. In the homomorphic setting (H), this mainly concerns the random factors $h^{w_i} \bmod n$ of the DGK system needed in the third step of the H comparison. In GC, the most intensive part of the oblivious transfers is precomputed (See Section 3.2.2). And for the secret sharing protocols, we pre-compute all multiplication triplets, and bitwise shared random numbers.

Figure 6.1 shows the CPU time (computation complexity) required for the pre-computations. The H approach outperforms all other approaches, while NO07 has the worst performance. The sudden drops and peaks in NO07 performance are due to the abort-probability explained in Section 5.3.2.

The communication complexity and bandwith shown in Figure 6.2 and Figure 6.3 show simmular patterns. Considering the amount of data transmitted during the pre-computation phase, the GC approach shows great results together with the homomorphic protocol, which both mainly consist of key exchanging. The secret sharing protocols CH10 and GSV07 enforce the transmission of a lot of multiplication

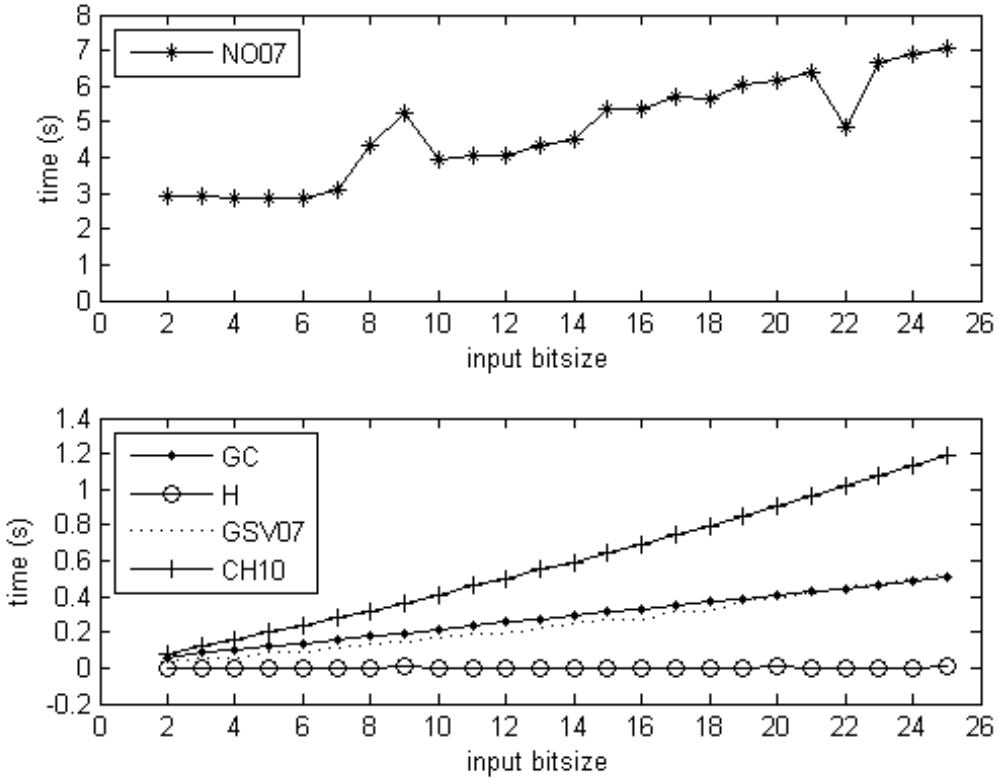

Figure 6.1: Time for precomputing

triplet data proportioned to the input size ℓ. The constant round NO07, again, does not fair very well due to the extreme high amount of multiplication triplets needed for this protocol.

6.3 On-Line Phase

The on-line performance can be found in Figure 6.4, Figure 6.5 and Figure 6.6. This time, the homomorphic protocol is outperformed by all other methods. But not by the same margin by which NO07 was outperformed during pre-computation. The increasing slope of the Homomorphic setting is explained by the increase in the number of e-values which have to be blinded by computing $[\![e_i]\!]^{m_i}$. Also the number of encrypted e_i's which need to be communicated increases linearly with the input size. The front spike of the GSV07 protocol in computation complexity (Figure 6.4) comes from the relatively rare event $[\varepsilon]_B = 1$. This scenario is computationally costly and occurs more often when p gets smaller. This spike in CPU time for small ℓ did not occur in NO07 because we kept p somewhat larger ($p \geq 2^7 - 1$) to prevent the abort-probability to escalate. Also, the p used in CH10 is large by construction: $\ell(p) = \kappa + \ell + 3$.

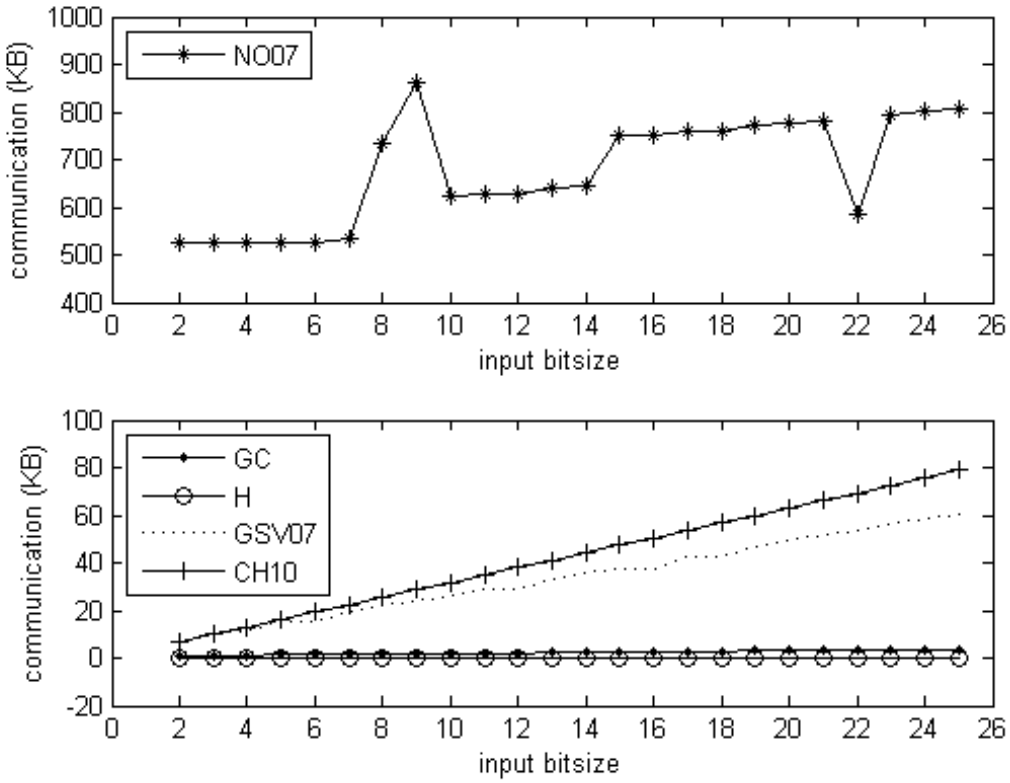

Figure 6.2: Amount of transmitted data for precomputing

6.4 General Performance

Figure 6.7 shows the total run-time of all protocols for an input of 25 bits. Consider the overall computation complexity, the homomorphic based protocol clearly performs best, but has a very dominant online computation demand of around 92 milliseconds per comparison.

As for the data performance for $\ell = 25$, Figure 6.8 shows that the GC based protocol has the best performance, while the homomorphic setting is still significantly better than the secret sharing based protocols. While all others have an overwhelming amount of communication during pre-computation (especially NO07), for the homomorphic protocol the main data transmission happens during the online phase.

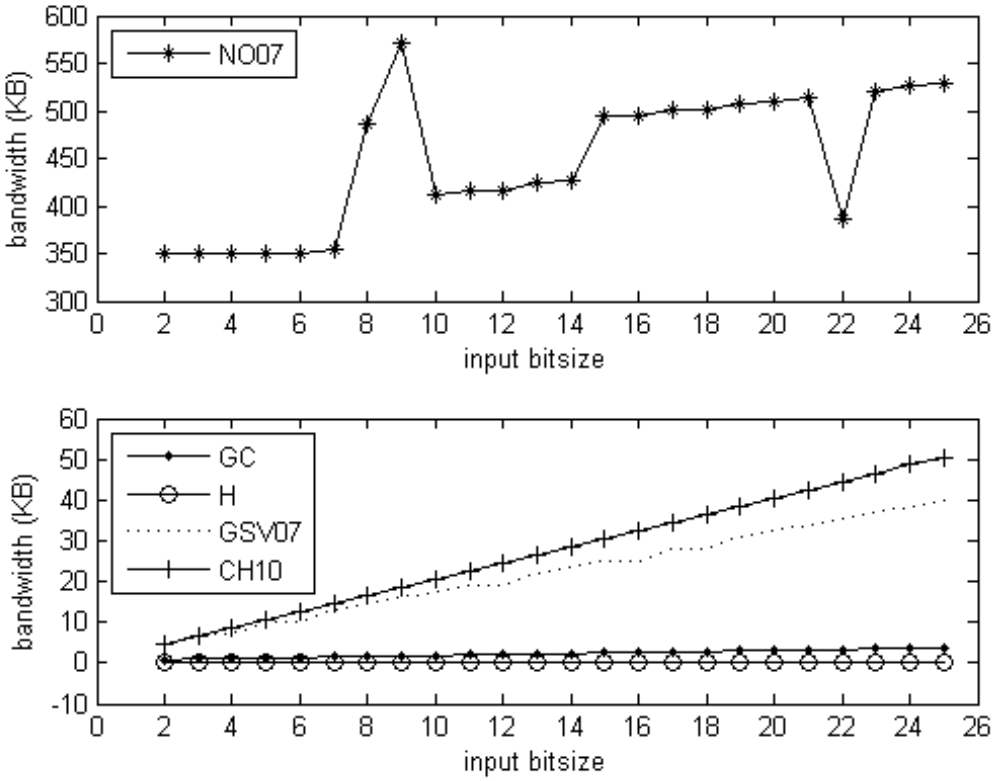

Figure 6.3: Required bandwidth for precomputing

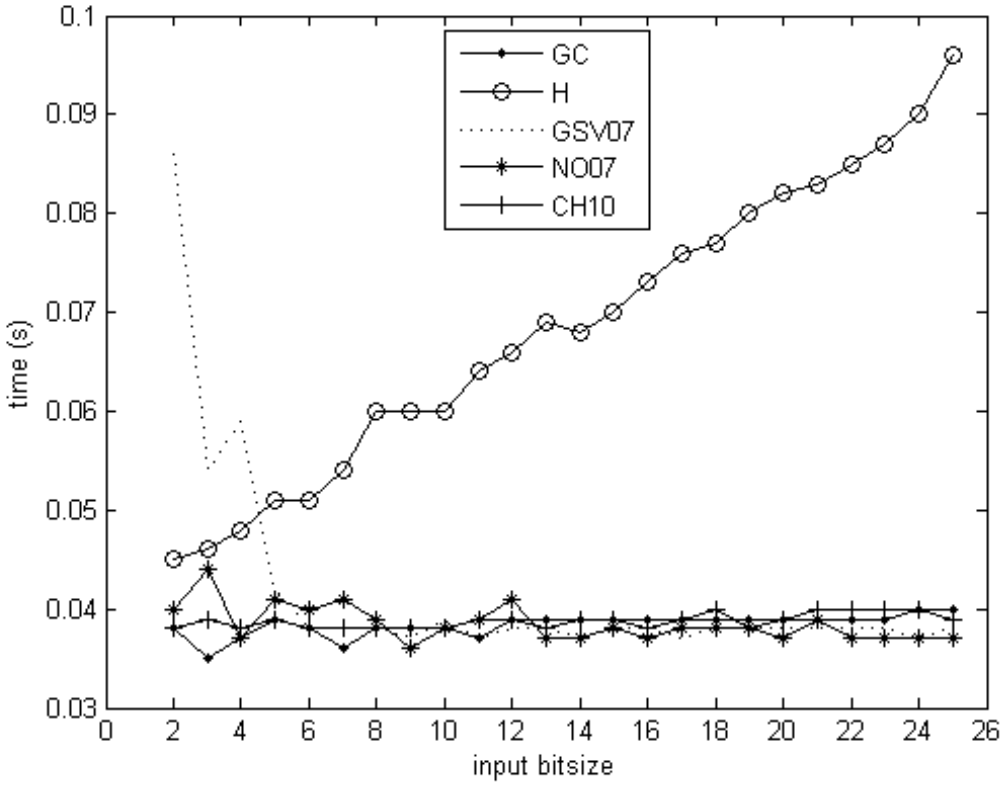

Figure 6.4: Time for online phase

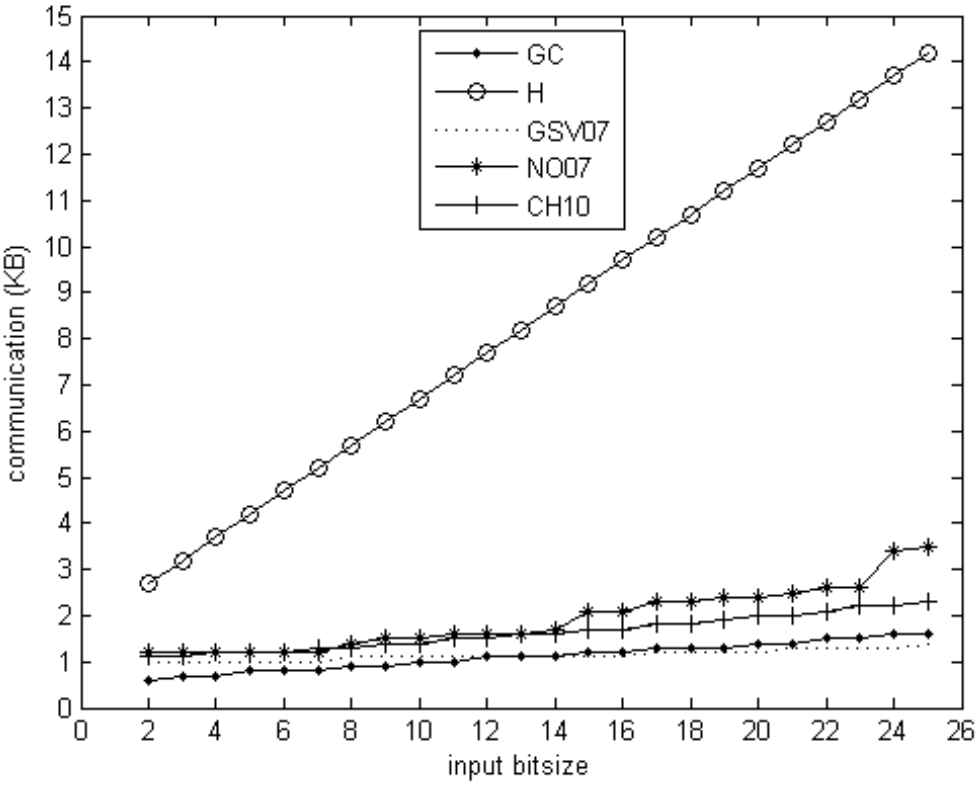

Figure 6.5: Amount of transmitted data for online phase

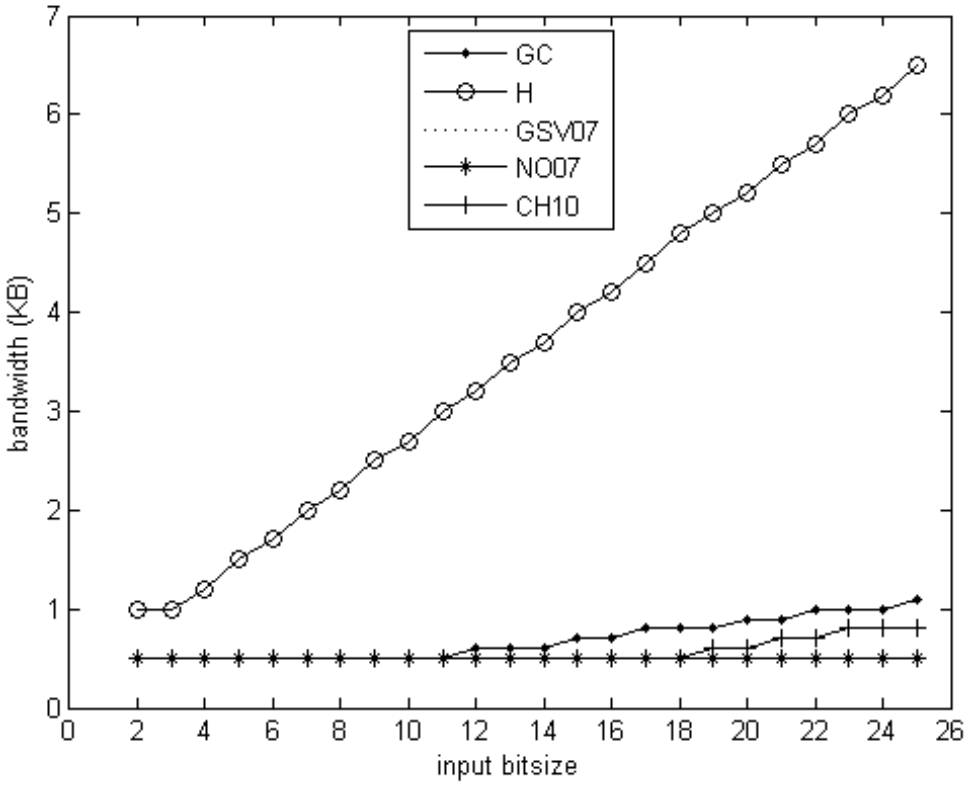

Figure 6.6: Required bandwidth for online phase

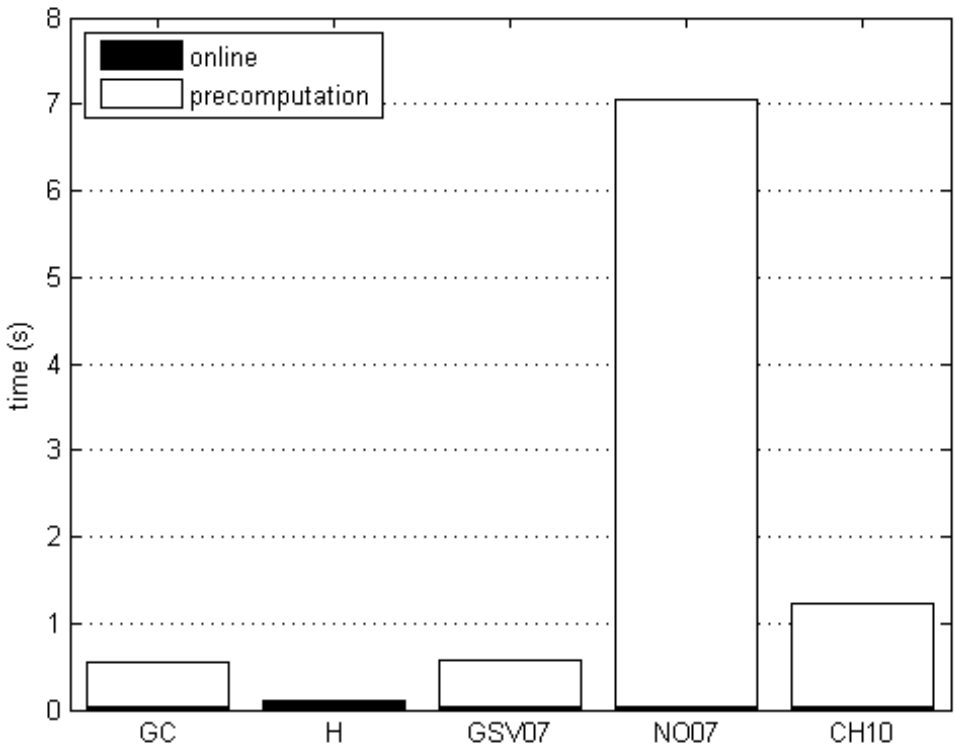

Figure 6.7: Total time

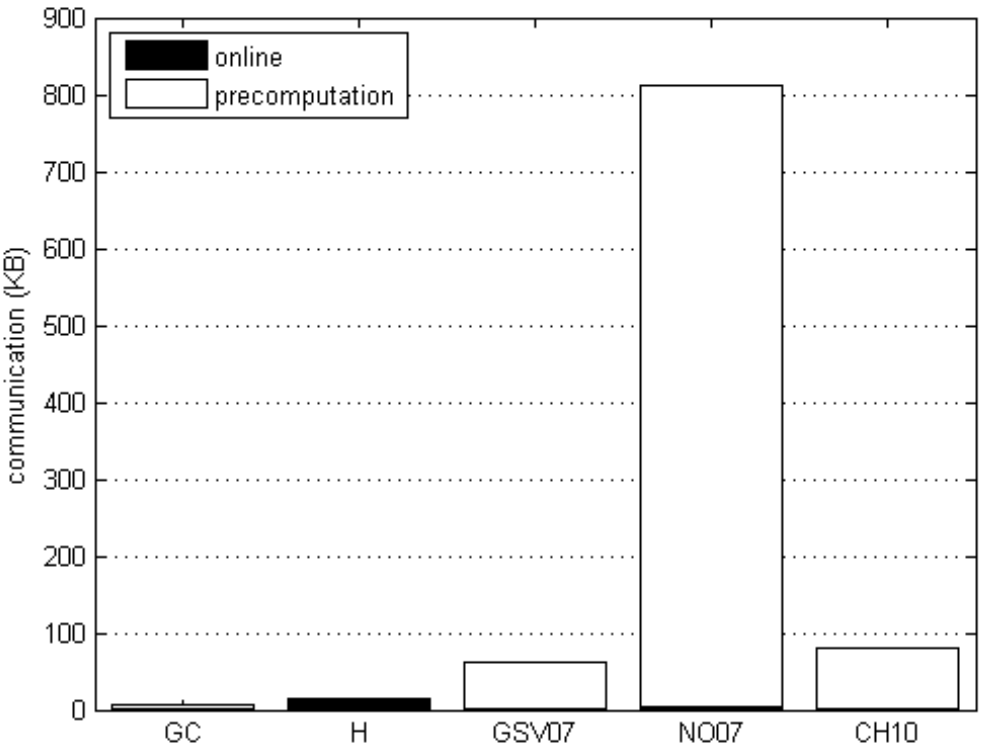

Figure 6.8: Total amount of transmitted data

Conclusion

For this research project, five different secure comparisons were compared by means of their computation, communication and round complexity as well as their imposed bandwidth. In order to generate relevant data to the broader research field of secure multi-party computation we introduced a general setting where Bob holds Paillier encrypted values en let the protocols terminate with Bob obtaining the Paillier encrypted comparison result.

Our experimental results show that, in the semi-honest model, the secret sharing protocol of Nishide and Ohta is not preferred for low input sizes. With their total of 28 communication rounds in our model, the promise of a secret sharing protocol with constant number of rounds and small p seems a bit bleak. The amount of send data and bandwidth required during pre-computations highly exceeds that of the other protocols.

As for the other two secret sharing algorithms by Garay, Schoenmakers, Villega and Catrina, de Hoogh, perform very well on the on-line phase, their performance is however lacking during the pre-computation phase.

The best performer on overall communication and round complexity turned out to be the Garbled Circuits with free XOR by Kolesnikov and Schneider as well as the homomorphic solution by Damgård, Geisler and Krøigaard. Of the total 4 rounds (aside from initialization) 2 can be precomputed in the GC setting against zero in homomorphic.

Where Garbled Circuit enjoys the best on-line performance, the homomorphic protocol turns out to be the better overall performer.

Bibliography

[1] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. IEEE Trans. on Knowl. and Data Eng., 17(6):734–749, 2005.

[2] C. Aguilar-Melchor, S. Fau, C. Fontaine, G. Gogniat, and R. Sirdey. Recent advances in homomorphic encryption: A possible future for signal processing in the encrypted domain. Signal Processing Magazine, IEEE, 30(2):108–117, 2013.

[3] J. Bar-Ilan and D. Beaver. Non-cryptographic fault-tolerant computing in a constant number of rounds of interaction. In Proc. 8th annual ACM Symposium on Principles of distributed computing, pages 201–209. ACM Press, 1989.

[4] E. Barker, W. Barker, W. Burr, W. Polk, M. Smid, P. D. Gallagher, and U. S. For. Nist special publication 800-57 recommendation for key management part 1: General, 2012.

[5] D. Beaver. Precomputing oblivious transfer. In Advances in Cryptology - CRYPTO95, volume 963 of Lecture Notes in Computer Science, pages 97–109. Springer, 1995.

[6] M. Bellare and S. Micali. Non-interactive oblivious transfer and applications. In Advances in Cryptology CRYPTO 89 Proceedings, volume 435 of Lecture Notes in Computer Science, pages 547–557. Springer New York, 1990.

[7] O. Catrina and S. de Hoogh. Secure multiparty linear programming using fixed-point arithmetic. In Proceedings of the 15th European conference on Research in computer security, ESORICS'10, pages 134–150, 2010.

[8] CBC. Federal agency loses data on 583,000 canadians. CBC News, 11 January 2013. Online, `http://www.cbc.ca/news/canada/federal-agency-loses-data-on-583-000-canadians-1.1327172`.

[9] J.-S. Coron, D. Naccache, and P. Paillier. Accelerating okamoto-uchiyama public-key cryptosystem. Electronics Letters, 35(4):291–292, Feb 1999.

[10] I. Damgård, M. Fitzi, E. Kiltz, J. Nielsen, and T. Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In Proc. of 3rd Theory of Cryptography Conference (TCC'06), volume 3876 of LNCS, pages 285–304. Springer-Verlag, 2006.

[11] I. Damgård, M. Geisler, and M. Krøigaard. A correction to 'efficient and secure comparison for on-line auctions'. IJACT, 1(4):323–324, 2009.

[12] J. Garay, B. Schoenmakers, and J. Villegas. Practical and secure solutions for integer comparison. In PKC # 07, volume 4450 of LNCS, pages 330–342, Berlin, 2007. Springer-Verlag.

[13] O. Goldreich. Foundations of Cryptography. Basic Applications, volume 2. Cambridge University Press, first edition, May 2004. ISBN 0-521-83084-2.

[14] S. d. Hoogh. Design of Large Scale Applications of Secure Multiparty Computation: Secure Linear Programming. PhD thesis, Eindhoven University of Technology, the Netherlands, July 2012.

[15] I. Damgård and M. Geisler and M. Krøigaard. Efficient and secure comparison for on-line auctions. In Information Security and Privacy, volume 4586 of Lecture Notes in Computer Science, pages 416–430. Springer Berlin Heidelberg, 2007.

[16] I. Damgård and M. Geisler and M. Krøigaard. Homomorphic encryption and secure comparison. Journal of applied cryptology, 1(1):22–31, 2008.

[17] V. Kolesnikov, A.-R. Sadeghi, and T. Schneider. Improved garbled circuit building blocks and applications to auctions and computing minima. In CANS, volume 5888 of Lecture Notes in Computer Science, pages 1–20. Springer-Verlag, 2009.

[18] V. Kolesnikov and T. Schneider. Improved garbled circuit: Free xor gates and applications. In ICALP (2), pages 486–498, 2008.

[19] Y. Luo, T. Pignata, R. Lazzeretti, S. ching Samson Cheung, and M. Barni. An efficient protocol for private iris-code matching by means of garbled circuits. In Proceedings of 19th International Conference on Image Processing, pages 2653–2656. IEEE, 2012.

[20] E. MacAskill. NSA paid millions to cover prism compliance costs for tech companies. The Guardian, 23 August 2013. Online, `http://www.theguardian.com/world/2013/aug/23/nsa-prism-costs-tech-companies-paid`.

[21] M. Naor and B. Pinkas. Efficient Oblivious Transfer Protocols. In Proc. SIAM Symposium on Discrete Algorithms (SODA 2001), Jan. 2001.

[22] T. Nishide and K. Ohta. Multiparty Computation for Interval, Equality, and Comparison Without Bit-Decomposition Protocol. In PKC 2007, volume 4450 of LNCS, pages 343–360. Springer-Verlag, 2007.

[23] T. Okamoto and S. Uchiyama. A new public-key cryptosystem as secure as factoring. In EUROCRYPT, pages 308–318, 1998.

[24] P. Paillier. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In EUROCRYPT 1999, volume 1592 of LNCS, pages 223–238. Springer-Verlag, 1999.

[25] N. Ramakrishnan, B. J. Keller, B. J. Mirza, A. Y. Grama, and G. Karypis. Privacy risks in recommender systems. IEEE Internet Computing, 5(6):54–62, 2001.

[26] T. Reistad and T. Toft. Linear, constant-rounds bit-decomposition. In ICISC, pages 245–257, 2009.

[27] A. Shamir. How to share a secret. In Communications of the ACM, 22(11), pages 612–613, 1979.

[28] V. Shoup. A Computational Introduction to Number Theory and Algebra. Cambridge University Press, 2nd edition, 2008.

[29] T. Toft. Primitives and Applications for Multi-party Computation. PhD dissertation, University of Aarhus, Denmark, BRICS, Department of Computer Science, 2007.

[30] T. Veugen. Encrypted integer division. In IEEE Workshop on Information Forensics and Security, Seattle, 12 2010. IEEE.

[31] A. C. Yao. Protocols for secure computations. In Proc. of the 23th IEEE Symposium on Foundations of Computer Science (FOCS '82), pages 160–164. IEEE Computer Society, 1982.

Appendix A

Implemented Constants

A.1 Sophie Germain Prime

The Sophie Germain prime used in our implementation of the $\binom{2}{1}$-OT protocol:

$$
\begin{aligned}
q = {} & 97785025993545014519724301465156051965563816095431483208285385505378481086154787879664339602874082829282020057084544383665319872770210279507247407549227685829168135418045682131933129\\
& 80424786601968345488641376108915233627538451906516884101127731738658639753626710116526129073196903012089565172250879871313126742177439414841230706501071977393334193050101128618629003863443184114012493526404222016400025058204497934780597604288323525\\
& 48050969690497022324535453077325171974437566562622021597002485828338293680772830745572932917086606046565984710034468130864831536419450159033516307582650066117745318192723929266741061794 1
\end{aligned}
$$

A.2 Lagrange Polynomial

The Lagrange polynomials $f_k \in \mathbb{Z}_p[X]$ of degree k used in our implementation of the Unbounded Fan-In Or:

$$f_1 = X - 1$$

$$f_2 = 2^{-1}X^2 + 5 \cdot 2^{-1}X - 2$$

$$f_3 = 6^{-1}X^3 - 3 \cdot 2^{-1}X^2 + 13 \cdot 3^{-1}X - 3$$

$$f_4 = -24^{-1}X^4 + 7 \cdot 12^{-1}X^3 - 71 \cdot 24^{-1}X^2 + 77 \cdot 12^{-1}X - 4$$

$$f_5 = 120^{-1}X^5 - 1 \cdot 6^{-1}X^4 + 31 \cdot 24^{-1}X^3 - 29 \cdot 6^{-1}X^2 + 87 \cdot 10^{-1}X - 5$$

$$f_6 = -720^{-1}X^6 + 3 \cdot 80^{-1}X^5 - 59 \cdot 144^{-1}X^4 + 37 \cdot 16^{-1}X^3 - 319 \cdot 45^{-1}X^2 + 223 \cdot 20^{-1}X - 6$$

A.3 Paillier Keys

The Paillier keys used in our implementation are the same throughout. The scheme is initiated as the 'short-cut' version ($g = n + 1$) via

$p = 177660192844068535117320188018234541393237728272787442847709125461839070411064421814516578038112854828373097896671184197454493969986337410838816875024415774945409277537031193361465140021530765862520563284698137745318339199802941525983688919126072346569877244493898614776456052047230386786420740701882961405323$

and

$q = 114731754603254775844465510064623293088931873558298480192432278966319365684771521802634756780817400417415900678892219814845312158956759792617758557270343191427256324139371069056178803015385815581081037752097593724834675984335472839233672103686248277171352825915938125616155721778632591847973785479824344817347$

Acknowledgements

This thesis is created as a final graduation product to my Master Mathematics at the VU University Amsterdam. The research it covers was carried out during my internship at TNO - Technical Sciences - ICT Security. So first of all, I would like to thank my daily supervisor at TNO Thijs Veugen and the ICT Security expertise manager Paul de Jager for offering me this great opportunity. On top of that I would like to thank Thijs Veugen for his incredible patience with me while I might imagine I often overwhelmed him with my many questions ranging from "How was your weekend?" to "How on earth could this possibly work?". He has given me the motivation and the support needed to function properly in the given office environment.

Also, my many thanks to all colleges I encountered during my internship. Whose help has made me feel welcome and appreciated in a comfortable working place. Special thanks on this matter are in order to Sebastiaan de Hoogh and Zekeriya Erkin from the Cyber Security Group at the Delft University of Technology, this work would not have existed as it is now without them.

I also want to address my gratefulness to my supervisor from the VU University Amsterdam Rob de Jeu and my second reader Sebastiaan de Hoogh for taking the time and effort to grade my work. I hope they are as pleased with this work as I am by the time they read this.

And last but not least, I wish to thank my girlfriend Suzanne van den Bosch who, despite this not being her field of expertise, listened to my endless explaining on cryptographic protocols and models. Because this helps me grasp the subject at hand and gives me confidence in my work.