An improved privacy-preserving clone detection mechanism against realistic adversaries

An Improved Privacy-preserving Clone Detection Mechanism against Realistic Adversaries

Master Thesis

Reto Weingart

September 8, 2010

Advisors: Prof. Dr. Srdjan Čapkun, Davide Zanetti

System Security Group

Department of Computer Science, ETH Zurich

Abstract

Recently, a novel privacy-preserving clone detection mechanism for RFID-enabled supply chains has been presented [1]. Since all collaborating partners in the supply chain are reluctant to share their private data, the protocol uses secure multi party computation to hide the actual inputs from the other players. However, the presented mechanism is secure in the semi-honest adversary model only. In order to use the protocol in a more realistic environment, we improved the mechanism to deal with a more powerful adversary model, which is called covert adversary model. The basic steps of the protocol stayed the same, but the used subprotocols had to be made secure against the new attacker model. We describe the covert adversary model, explain the newly used subprotocols and evaluate the performance of the developed protocol.

Contents

List of Figures

CHAPTER 1

Introduction

In this chapter, we explain a RFID-enabled supply chain, the possibility to inject counterfeit products into such chains and methods to detect and evaluate cloning.

1.1 RFID-enabled Supply Chains

Nowadays more and more products are equipped with a RFID tag. The tags are cheap to produce, can store a lot of information compared to their small size and can easily read out by machines. For this reason, they are often used in supply chains of products to keep track of the individual items, to get information of the flow in the chain, to increase automation and optimize the capacity and load of the stocks.

Supply Chains

A supply chain is the way of a product from its creation by the supplier to its sale. In between, there can be a lot of companies and persons involved, which are called partners of the supply chain.
In this work, we are interested in how the products flow between the partners and especially the data the partners get out of the monitoring of the RFID tags.

RFID

RFID is an abbreviation for Radio Frequency Identification, a technique to read out stored data from an object using radio waves. There are basically three different types of RFID tags. The active ones, which contain a battery and are able to send signals autonomously. The passive RFID tags do not have a battery and therefore rely on an external source to transmit a signal. The third type is somehow a hybrid type, which has a battery but needs an external signal to transmit though.
The reader generates an electromagnetic field that is received by the RFID antenna and gets converted into energy, which is then used to send back the stored informations.
In the supply chains, the products get read out on every receive or send of a partner. Basically, the RFID tag of a product in a supply chain contains its unique identifier. This is a number, which is unique for every product in the chain, used to distinguish them.
The partners of the supply chain are reluctant to share the recorded data with the other partners, because it is possible to get sensitive information out of the data about their strategies, business relationships or target markets. With these informations, the others can improve their benefits or change the strategy, which then weakens the other player. Due to this fact, all the partners are interested in maintaining privacy of the data.

1.2 Cloning in RFID-enabled Supply Chains

Counterfeit products are getting more and more a problem in our world. Expensive and popular products get copied and are sold to much lower prices and mostly with much lower quality too. The financial loss and the damage of reputation can be very high for the producer of the genuine products. For this reason, the genuine producing companies want to have technical and judicial protections against counterfeiting.
Depending on the product, counterfeits are more or less difficult to produce or to detect. Physical products like clothes or watches are much more difficult to counterfeit than electronic products like electronic documents or pictures. Cloning a picture on a electronic device does not need special knowledge. The detection, on the other hand, has somehow the inverse difficulty. To detect a cheap counterfeit of an expensive watch is not very difficult for experts. However, the distinction between two pictures on an electronic devise is difficult and sometimes even impossible.
To successfully counterfeiting products with a RFID tag, it is firstly required to make an exact copy of the physical product and secondly to clone its RFID tag. This is certainly doable since getting a valid ID for a product can easily be obtained from the EPCglobal registry [4].
After the creation of the counterfeit, it is injected into the supply chain by

fraudulently selling it as a genuine product to a partner. Once being in the supply chain, the counterfeit is shipped along the chain and is sold to a customer at the end. So it is possible that even a honest partner gets a counterfeit and sends it to the next partner. But due to the possible damage in reputation or the financial loss in the case of a complaint when a counterfeit is detected, the partners are interested in detecting the clones on the reception of the products and not sending them further on to the next partner.
Since the physical inspection of all the incoming products by humans takes a long time and therefore is very expensive, it is desired to make the detection by machines on the RFID tags. For this to work, we have to be able to detect if the RFID tag of a counterfeit is the clone of a legitimate product's tag.

1.3 Preventing and Detecting Clones

There are basically two approaches to work against counterfeiting, the clone prevention methods and the clone detection methods.
The purpose of clone prevention methods is to make cloning of the RFID tag very hard. For this case cryptographic primitives like digital signatures or encryptions are used. This approach is also called *tag-based*. But this solution comes with a price. There are extra resources needed to check the signatures or take care of the key management.
In this work, we focus on the clone detection approach, often denoted as *location-based* method. The observed data of the RFID tags recorded by the partners of the supply chain are used for the analysis. Since every product has a unique ID on the RFID tag, the partners store an observation tuple for every activity like reception and sending. As soon as a product is received by a partner, the timestamp T, location L and direction D is recorded for the observed ID.

$$(ID, T, L, D)$$

The direction D indicates whether the observation took place on the reception from the previous partner ($D = RCV$) or on the sending to the next partner ($D = SHP$) of the item. All these observations are stored in the partners' local databases. For the execution of the clone detection, all the partners' observations must somehow be collected and time-sorted for all present identifiers. At the end a time-sorted list of observations for all products is obtained. For every two time consecutive events e_i and e_{i+1} in this list for a given ID, one can make a statement about clones in the supply chain by applying two simple rules:

$$R_1 : e_i = (ID, *, L_j, RCV) \rightarrow e_{i+1} = (ID, *, L_j, SHP) \tag{1.1}$$
$$R_2 : e_i = (ID, *, L_j, SHP) \rightarrow e_{i+1} = (ID, *, L_o, RCV) \tag{1.2}$$

Rule R_1 says that, for a given tag, a receiving event (RCV) recorded at location L_j must be followed by a shipping event (SHP) recorded at the same location, while rule R_2 says that, for a given tag, a shipping event recorded at location L_j must be followed by a receiving event recorded at a different location L_o. [2]
Theoretically, if one of the above rules do not hold, one can conclude directly that an item with a cloned legitimate RFID tag is in the chain and therefore cloning is detected. This might not always be true, since tag misreads can occur and therefore will influence the rule verifications. So after evaluating the rules for all observations, the partners are able to individually decide if there exists counterfeits according to their own threshold values τ. They compute the ratio R_{RV} between the number of failed rule verifications VR_F and the total number of verified rules VR. If then $R_{RV} < \tau$, the partner assumes cloning.
However the partners do not want to reveal their locally stored observations in order to run a clone detection protocol, the detection mechanism has to guarantee privacy preservation. This is the point where the secure multi party computation and the other cryptographic primitives come into play. A detailed description and explanation of these techniques can be found in Chapter 3.

CHAPTER 2

Problem Statement

Due to possible misreads which can lead to either a false failed rule verification or a false passed one, in [1] another measurement for the analysis was proposed. The new method assumes a binomial distributed occurrence of the misreads with a failure probability p_{mr}. The measurement tests the deviation from the expected number of failed rule verifications caused by misreads, considering the given distribution. Like in the rule verification based approach, clones are detected if the computed number exceeds a certain threshold.

2.1 BT Clone Detection

The BT clone detection method uses the binomial test, where the null hypothesis H_0 says, that the failed rule verifications are caused by misreads, while the alternative hypothesis H_A says, that those are caused by clones:

$$Pr(K \geq k) = \sum_{k=N_{ME}}^{N_E} \binom{N_E}{k} p_{mr}^k (1 - p_{mr})^{N_E - k}$$

$$Pr(K \geq k) \leq \alpha \rightarrow H_0$$

$$Pr(K \geq k) > \alpha \rightarrow H_A$$

where N_{ME} denotes the total number of missing events, that are needed to correctly restore the sequence of observations for all products, to pass the rules. The number N_E denotes the total number of events and α is the significance level of the test. Table 2.1 lists the possible rule failures and the

Figure 2.1: Relations between failed rule verifications and missing events

L_i	D_i	L_j	D_j	Failed rule	Missing events ($\#_{ME}$)
A	RCV	A	RCV	R_1	A-SHP, B-RCV, B-SHP (3)
A	RCV	B	RCV	R_1	A-SHP (1)
A	RCV	B	SHP	R_1	A-SHP, B-RCV (2)
A	SHP	A	RCV	R_2	B-RCV, B-SHP (2)
A	SHP	A	SHP	R_2	B-RCV, B-SHP, A-RCV (3)
A	SHP	B	SHP	R_2	B-RCV (1)

corresponding missing events.

In [1] a privacy-preserving version of the BT clone detection in RFID-enabled supply chains was presented. The protocol uses some cryptographic tools to preserve the privacy of the partners' recorded data. It works in a scenario where n partners P_i own N_{TE} tag events e_j for a product with a given ID. They guarantee, that during the execution of the protocol, the partners will not learn anything about the other partners' events and relationships, and neither about the structure of the supply chain. The privacy issue prevents other partners from learning something about the events itself, like event e_j follows event e_i in time or informations about the order of partners occurring in the chain.
The protocol guarantees security in the semi-honest adversary model.

Semi-honest Adversary

The semi-honest adversary is a passive model. A semi-honest player follows exactly the protocol instructions, but may try to learn more information than just his output. The adversary tries to analyze the transcript of messages received during the execution of the protocol. This is just a small deviation from the honest adversary, who sticks to the protocol definition too, but has no intention to learn more information than specified in the protocol. Security against semi-honest adversaries is therefore a very weak definition and not that realistic for real world applications.

2.1.1 The Existing Protocol

The proposed protocol for the clone detection first verifies the rules (1.1) and (1.2) in Section 1.3 for every pair of events, in order to evaluate the number of missing events between them. After that, all pairs of time-consecutive events have to be identified and summed up the number of missing events between these pairs to get the total number of missing events N_{ME} in the chain.
The brief description of the proposed protocol can be found in [2] and the

corresponding description of the new protocol is written in Section 4.5.

2.2 Weaknesses of the Existing Protocol

The security related weakness of the protocol, is the assumption of the semi-honest adversary model. This model might have some interests in the theoretic world though, but is not really applicable in the real world. In an environment where money exists and the pursuit of success and power is on everybody's mind, this model is not realistic. In fact one can not expect, that somebody plays fair just for goodwill, but rather it has to be made the assumption, that a priori everybody wants to improve his benefits as the first goal and therefore will cheat if he has the possibility for that.
So the goal for the improvements is to change the adversary model to a more realistic one and adapt all the used subprotocols accordingly, in order to be secure in the new model.
To make the protocol secure in a new model, all used subprotocols need to be secure in this model. In Chapter 4, a listing can be found of the subprotocols and for each a brief description of weaknesses in terms of privacy and correctness, when an active adversary is considered instead of the semi-honest one.

2.3 Realistic Adversary Models

To meet the requirements to guarantee security and privacy in a more realistic environment than in the semi-honest adversary model, the security model should move from passive adversaries to some kind of active adversaries at least. However in [5, 6] has been shown, that under suitable cryptographic assumptions, any multi party probabilistic polynomial-time functionality can be securely computed for any number of active corrupted parties.
Of course this is not for free, the computational complexity is often beyond what would be feasible for actual used protocols in practice. For this reason, Aumann and Lindell [3] introduced a new adversary model that lies between the semi-honest and the active adversary definition. The goal was to have an adversary model, which is more realistic than the semi-honest and less complex than the active one. The protocols that deal with the new model should be feasible in terms of computational complexity and usable in practice without difficulty.

2.3.1 Covert Adversary

The more realistic adversary model, which is better suited in real world scenarios, is the so called covert adversary model, that lies between the semi-honest and active model. The adversary can deviate arbitrarily from

the protocol in an attempt to cheat, but does not wish to be "caught" doing so. Being caught cheating will result in loss of reputation and afford the embarrassment. In the covert adversary model, a cheating partner is detected with at least a certain probability, called the *deterrence factor* ϵ. According to the specific circumstances, this probability can be tuned to the desired value by different strategies.

The definition of the security is based on the classical *ideal/real simulation paradigm,*[1] and provides the guarantee, that if the adversary cheats, then it will be caught by the honest parties (with some probability ϵ).
Successful cheating means that an adversary manages to do something that is impossible in the ideal model. Successful cheating then, is any behavior, that cannot simulated in the ideal model.
For the deterrence factor ϵ, the ideal-model simulator is allowed to sometimes "fail" (meaning that the output distribution of the real protocol execution cannot be simulated in the standard ideal model for secure computation), with the requirement that in a real execution with the adversary the honest parties would detect cheating with probability, which is at least ϵ times the probability that the simulator fails.

The security guarantee does not preclude successful cheating. Indeed, if the adversary decides to cheat, then it may gain access to the other parties' private informations (depending on the formulation: failed-simulation, explicit-cheat or strong explicit-cheat[2]) or bias the result of the computation. *The absolute privacy and security is therefore not guaranteed, unlike in standard definitions for other adversary models like semi-honest or active adversaries.* In the explicit-cheat formulation, the adversary gets the honest parties' inputs anyway whenever he wants to cheat and in the strong explicit-cheat formulation, he just gets the inputs in case the cheating is not detected, which has the probability $1 - \epsilon$.
The absolute correctness is not guaranteed either, because in a successful attempt to cheat, the adversary defines what will be the output of the honest parties (for both the explicit-cheat and strong explicit-cheat formulation). In case the attempt to cheat is not successful, the honest parties receive the output $corrupted_i$ from the trusted third party and the protocol ends.

[1] In this paradigm, the execution of a real protocol is compared with a ideal execution where a trusted third party receives the parties' inputs, computes the function and returns the outputs. Formally, a protocol is secure if for every real-model adversary A attacking the protocol there exists an ideal-model adversary/simulator S (with the trusted third party) such that the output distribution of the honest parties and S in an ideal execution is computationally indistinguishable from the output distribution of the honest parties and A in a real execution of the protocol.

[2] see Appendix A.1 for the definitions

The adversary can always abort the protocol which means, that all the honest parties get the output $abort_i$. That is not a problem, since a crashing honest party can just store his input and random-tape on disk before the execution begins. Then, before sending any message, the incoming messages prior received are also written to disk. The result of this is, that whenever a party's machine crashes, it can easily reboot and return to its previous state. We therefore believe that honest parties cannot truly hide behind the excuse that their machine crashed (it would be highly suspicious that someone's machine crashed in an irreversible way that also destroyed their disk at the critical point of a secure protocol execution).
The security parameters for protocols in the covert adversary model can be set in a way, that the deterrent factor comes arbitrarily close to 1, but not negligible close to 1 though. But every improvement of ϵ comes with a price in terms of the complexity and running time of the protocol.

2.3.2 Active Adversary

The active or malicious adversary model is the strongest possible definition for corrupted parties. They may behave arbitrarily and can derive from the protocol specification in every thinkable way. They can decide to send wrong values, not expected messages to other parties, drop messages or even stop sending anything in order to appear offline for other players. If a protocol is secure in the active adversary model, it is secure in the covert and semi-honest model too.
The mentioned strong explicit-cheat formulation[3] of the covert adversary definition "converges" to the malicious model as ϵ approaches 1.

2.4 Motivation for the Covert Adversary Model

Some keywords about the motivation for using the covert adversary as the security model has already been mentioned in Section 2.3. A new model is required that defines security in a reasonable manner, which is applicable for realistic scenarios, but does not have the huge complexity like protocols providing active security. The reason why this model fits the real world scenarios better, is because it takes the partners' willingness to cheat into account. Parties are willing to actively cheat only in the case they are not caught. This behavior is the same as in the real business world, where a caught cheating person cannot afford the embarrassment, loss of reputation, and negative press which is brought along with the detection.

[3] see Appendix A.1 for the definition

CHAPTER 3

Security Solutions – Secure Multi Party Computation for Covert Adversaries

The essential part in a privacy-preseving clone detection protocol is the secure multi party computation (SMC). This cryptographic primitive allows multiple parties to jointly compute a function of their inputs in a way that every party gets the correct output (correctness) and no one learns something other than the output, especially nothing about the other players' inputs (privacy).
However in [5, 6, 8, 9, 10] it has been shown, under appropriate cryptographic assumptions, that multi party computation of a polynomial-time function is doable for any number of active adversaries, the price for this is very high. For most desired applications, such a protocol is not feasible. Due to this fact, in [3, 7] new multi party computation protocols has been presented for the covert adversary model, which still has a relatively high security guarantee but at a much lower price than the active secure ones.
The idea behind their solutions is to employ the cut-and-choose technique to make the protocols secure against covert adversaries.

3.1 The Cut-and-choose Technique

Cut-and-choose is a often used technique to enhance the security for protocols working in the semi-honest adversary model. The basic idea, loosely

spoken, is that in a first step one player creates l instances of the semi-honest protocol and the other players come up with a number $i, 0 \leq i < l$. Then the first player has to reveal all used information to generate the instances $I_j, 0 \leq j < l$ except for the instance I_i. As a next step, the other players can check if these instances were constructed correctly and compute what was specified. With the not opened instance I_i, the protocols proceed the execution.
This technique is applied whenever the players have to compute or generate security-critical values, where cheating possibly can affect the correctness or privacy of the protocol.
The parameter l directly defines the security measure of the final protocol. In order to successfully cheat without being caught, the cheating player has to choose the same instance for cheating as the other players will define for the continuation. In all other instances the cheater has to execute exactly the steps defined in the protocol, otherwise he will get caught by the other players when revealing the used corrupted data in the verification phase.
The probability for the cheating player to predict the right index for the corrupted instance is $\frac{1}{l}$. Only with this probability he manages to cheat without being noticed by the others. As a result, a cheating player gets caught at least with probability $1 - \frac{1}{l}$, which is called the deterrence factor ϵ of the covert adversary model definition.
Depending on the required security level, this parameter l can be set to achieve an arbitrarily value of the deterrence factor between 0.5 and 1, even though it is impossible to get negligible close to the probability 1.

3.2 SMC for Covert Adversaries

As already mentioned several times, the secure multi party computation is very essential for the presented privacy-preserving clone detection protocol. One of the main subprotocol used is the computation of Yao circuits (see Section 4.3), which builds up on the oblivious transfer (see Section 4.2) primitive. In both protocols, the cut-and-choose technique is used to improve the security guarantee from the semi-honest to the covert adversary model. The detailed functionality and the security aspects are explained in the next chapter.

CHAPTER 4

The Protocol for Covert Adversaries

In this chapter the actual privacy-preserving clone detection protocol against covert adversaries is presented. First of all, the subprotocols are described in detail explaining the idea, the security flaws of the semi-honest version, the steps of the protocol and the intended purpose they are used for.
The security analysis of all the subprotocols as well as the whole protocol is done in Chapter 5.

4.1 Distributed Key Generation

In order to hide the inputs and intermediate results from the partners, the values have to be encrypted. To meet the requirements stated in Section 4.1.1, a public-key cryptography scheme is used. The used public key/secret key pair for this scheme has to be generated and sent to all partners participating in the clone detection protocol. This is exactly the functionality of the distributed key generation (DKG) subprotocol.

4.1.1 DKG Definition and Former Works

In many security related applications, it is required that the secret key is not possessed only by a single party. That further means, that no single party can decrypt a ciphertext on its own. When dealing with active adversaries, it is needed that the secret key is somehow shared in m pieces among all the participating partners and that only a subset of size t can reconstruct it. A

group with less than t players has no information about the secret key. This cryptographic primitive is called a *t-out-of-m* secret sharing scheme.

Secret Sharing Scheme: Shamir sharing

A widely used and well known secret sharing scheme is the one presented by Shamir in [15]. The main idea is to use a polynomial $P(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{t-1} x^{t-1}$ of degree $t-1$, to share the secret s. Shamir's approach is based on the fundamental observation in analysis, that t points are sufficient to uniquely interpolate a polynomial of degree $t-1$.
To share a secret s in a *t-out-of-m* scheme, the first coefficient a_0 of the polynomial is set to s and all the other coefficients $a_1, \ldots, a_{t-1}$ are chosen uniformly at random. After the definition of the polynomial P, the m shares $s_i, i \in \{1, \ldots, m\}$ are computed as P evaluated at the points i, $s_i = P(i)$.
In order to reconstruct the secret s, t players must collaboratively interpolate the polynomial based on their points received in the sharing. The interpolation can be done with the Lagrange method. If the polynomial is interpolated, the secret s is obtained by evaluating P at position $x = 0$, $P(0) = s$.

A DKG mechanism that securely generates a secret key/public key pair and distributes the public key and a *t-out-of-m* sharing of the secret key to every player, is called a threshold distributed key generation scheme. Such schemes has been studied in many papers and were solved the first time in [11, 12] for a honest majority. Another protocol providing the threshold DKG functionality, was a protocol based on the threshold cryptosystem proposed in [13] by Pedersen. It uses the discrete logarithm problem for the security proof. However a few years later, Gennaro et al. [14] found a flaw in the security scheme which enables to influence the distribution of the key in the key space to not being uniform anymore. Therefore, the security of this protocol is not guaranteed anymore in the presence of a malicious adversary.

4.1.2 DKG against Covert Adversaries

For the new protocol with security against covert adversaries, the threshold distributed key generation scheme from Fouque and Stern [16] based on discrete logarithms is used. The scheme is an improvement of the Pederson scheme to deal with covert adversaries instead of just passive adversaries. The protocol uses the Paillier's cryptosystem[1] for encryption and needs just one round. No pairwise secure channels are needed for the protocol either.

[1] A probabilistic asymmetric algorithm for public key cryptography, see Section A.2 for further details

The only requirement is that broadcast channels are available and the network has to be synchronous.
To deal with active adversaries, Publicly Verifiable Secret Sharing (PVSS) and Publicly Verifiable Encryption (PVE) schemes are added to the protocol. These two mechanisms are used to determine players not following the protocol. The detected cheating players are disqualified and only the set of all honest players will then define the public key. Therewith it is also guaranteed that the players have correct secret key shares distributed.
For the detailed step by step description of the protocol, see Section B.1 in the appendix.

Publicly Verifiable Secret Sharing (PVSS)

For a secret sharing scheme to be publicly verifiable, two criteria must be satisfied. First, the scheme needs to be verifiable, meaning that auxiliary information is included, which allows players to verify their shares as consistent. The second criteria is that in the scheme, each party involved can verify the validity of the shares distributed by the dealer.
So the characteristic of the PVSS enables everybody's ability to check that a malicious adversary would not deviate from the protocol and does not influence the distribution of the secret key respectively.
In our new protocol, the PVSS scheme is used by every party to share their random number with. For the full explanation and usage of the PVSS we refer to [16].

Publicly Verifiable Encryption (PVE)

A publicly verifiable encryption scheme [18] in its basic form, is a two-party protocol between a prover P and a verifier V. Their common inputs are a public encryption key E, a public value x, and a binary relation R. As a result of the protocol, V either rejects or obtains the encryption of some value w under E such that $(x, w) \in R$ holds. In our new clone detection protocol, R is defined such that $(x, w) \in R$, if and only if w is the discrete logarithm of x to the base g. The PVE scheme is used to transfer the share to the intended party in such a way, that all parties can verify that the receiver is able to recover his share.
For the whole description of the PVE we refer to [16, 18].

4.2 Oblivious Transfer

Oblivious transfer (OT) is a primitive, that allows the receiver R to retrieve some information offered among several alternatives by the sender S, without S learning the choice of R. It was presented the first time in [19] by Michael O. Rabin.

An often used form of the protocol, is the *1-out-of-2 OT,* where the sending party S inputs two messages (m_0, m_1) and the receiving party R has a selection bit $\sigma \in \{0, 1\}$ and gets the message m_σ after the protocol execution. The privacy issues are that S has no information which message was chosen by the other party (privacy for R), and R just learns the chosen message m_σ and gets no information about the other message $m_{1-\sigma}$ (privacy for S). The OT is embedded in the Yao subprotocol used for the circuit evaluating party to get the input keys for the circuit, which correspond to its input bits. Since for every input bit of the Yao circuit, one OT has to be run, it is the subprotocol which is executed the most.

4.2.1 Homomorphic Encryption Schemes

Homomorphism is a characteristic for encryption schemes, which is used in many cryptographic applications. An encryption scheme is homomorphic if given two ciphertexts $c_1 = E_{pk}(m_1)$ and $c_2 = E_{pk}(m_2)$, encrypted with the public key pk, it is possible to efficiently compute $E_{pk}(m_1 + m_2)$ without knowledge of the secret decryption key. In order to be able to do that, the message space needs to be a group. In fact, we assume the message and ciphertext space to be a group with the group operations $+$ and $\cdot$ respectively. More formally spoken, the definition is the following:

A public-key encryption scheme is homomorphic, if for $m_1, m_2 \in \mathcal{M}$ holds that

$$\{pk, c_1 = E_{pk}(m_1), c_1 \cdot E_{pk}(m_2)\} \equiv \{pk, E_{pk}(m_1), E_{pk}(m_1 + m_2)\}.$$

The Paillier encryption scheme has exactly this property. It is an additively homomorphic encryption scheme as the following computation shows:

$$E(m_1) \cdot E(m_2) = (g^{m_1} r_1^z)(g^{m_2} r_2^z) = g^{m_1+m_2}(r_1 r_2)^z = E(m_1 + m_2 \mod z)$$

where z is the modulus, g the base, r_i the random coin for encrypting the message m_i and the encryption is defined as $E(m) = g^m r^z \mod z^2$.

4.2.2 OT against Covert Adversaries

Although there are OT protocols developed against malicious adversaries, the performance and bandwidth usage is often not feasible for practical applications. For these reasons, Lindell et al. [3] developed a new OT protocol, which is secure in the covert adversary model and outperforms the existing protocols significantly. It uses homomorphic encryption as a building block and takes a simple implementation of a semi-honest secure 1-out-of-2 OT as an underlying protocol.
In this section, we just briefly mention the key ideas of Lindell's protocol, a

detailed description can be found in Section B.2 of the appendix.

The basic idea of the underlying OT protocol is that the receiving party R generates a key pair for a homomorphic encryption scheme and encrypts a random bit α and the inverse $1-\alpha$ to ciphertexts c^0 and c^1 respectively. Depending on his selection bit σ, R sends the ciphertexts as a pair (c^1, c^0) for the case $\sigma = 0$, or as (c^0, c^1) for $\sigma = 1$ to the sending party S. S then uses the homomorphic property, the two ciphertexts and his two input messages (x_0, x_1) to compute the values for R as $\tilde{c}_0 = x_0 \cdot c_0$ and $\tilde{c}_1 = x_1 \cdot c_1$. If $\sigma = 0$, R computes the decryption of $\tilde{c}_0$, otherwise the decryption of $\tilde{c}_1$.
Due to the homomorphic property of the encryption, the not selected ciphertexts $\tilde{c}_{1-\sigma}$ is an encryption of the value zero, because the message $x_{1-\sigma}$ is multiplied by an encryption of 0 and gives therefore no information about the sender's input message corresponding to this bit $1-\sigma$. On the other hand, the ciphertext $\tilde{c}_\sigma$ remains the encryption of the sent message x_σ, because it was just multiplied by an encryption of 1 and therefore is not changed.

Due to some possible attacks on this protocol, as further discussed in Section 5.1.2, the protocol has to be enhanced by some strategies. In order to make the protocol secure in the covert adversary model, the already described cut-and-choose technique is employed.
To be precise, this technique is used twice. Once to check the validity of the key generation, referred as the *key-generation challenge*, and once to guarantee that the receiver does not use twice an encryption of one for the α and $1-\alpha$, referred as the *encryption-generation challenge*. For these two cases, R has to generate k key pairs (for the key-generation challenge) and k $(\alpha, 1-\alpha)$ pairs (for the encryption-generation challenge). After each of these two steps, the sending party S randomly selects and sends a bit b to R, who then needs to reveal the randomness used for generating all the key pairs and α pairs respectively, except the one with index b. The pairs with index b are the one used for continuation of the protocol.
In order for a corrupted receiver to cheat, he has to predict the choice b of the sender and must stick to the protocol for all the other indices. Since b is uniformly chosen at random out of k possible values $[1, \ldots, k]$, the probability of guessing the right index is $\frac{1}{k}$. The probability of getting caught after cheating is therefore $1 - \frac{1}{k}$, which is denoted as the deterrence factor ϵ of the OT subprotocol.

This approach uses the advantage of the simple protocol for a 1-out-of-2 OT and makes it secure against covert adversaries with the aid of the cut-and-choose technique. The used encryption scheme is the Paillier's encryp-

tion scheme[2] having the essential homomorphic property which is required.

4.3 Yao Circuits

The computation of Yao circuits is probably the most central point in our privacy-preserving clone detection protocol. In order to guarantee privacy of the partners' inputs, the both basic functions GT (greater-than) and ME (number of missing events), need to be computed in a secure manner. The protocol's requirements are that two partners can compare for each pair of observations, which event occurred before the other in time and the number of missing events between them.

The Yao scheme, introduced by Andrew Chi-Chih Yao in [10], is used to securely compute any given function between players in a way, that the result can be computed according to everyone's input, but nobody sees the other's inputs. The function to compute, is represented as a boolean circuit C with all the input and output wires and a truth table for each gate. In the protocol a two-player version of Yao is used.
There are two boolean circuits defined, one for computing the GT operation C_{GT}, the other for the ME operation C_{ME}. The circuit C_{GT} takes two 32-bit strings as input, representing the timestamp of the events, and outputs a single bit on three wires, representing the result greater-than (GT), less-than (LT) and equality (EQ). For example, for two events e_i and e_j with corresponding timestamps T_i and T_j where $T_i > T_j$, the output assignment of the three wires is the bit value 1 for the GT output wire, and twice the bit value 0 for the LT and EQ output wires. Although the output wires LT and EQ are not needed for the GT operation, they are needed for the ME operation.
The ME circuit C_{ME} takes two 33-bit strings as input, where 32 bits represent the location of the event and one bit the direction. The circuit has three output wires representing the number of missing events between the two observations, for one (ow_1), two (ow_2) or three missing events (ow_3) respectively. As an example, again for two events e_i and e_j with corresponding locations L_i and L_j where $L_i \neq L_j$ and directions $D_i = RCV$ and $D_j = SHP$, the number of missing events between them is 2, as it can be seen from Table 2.1 in Chapter 2. The output assignment for this computation would be the bit value 0 for the output wires ow_1 and ow_3 and the bit value 1 for the output wire ow_2. For two observations with no missing events in between, all the output wires will have the bit value 0 assigned.
In order to compare the locations of the two events, the previously described GT circuit is part of the ME circuit. There the output wire EQ of the GT circuit is used for the locations and directions, to determine whether they

[2] See Section A.2 for further details

Figure 4.1: Truth table for the binary relation AND

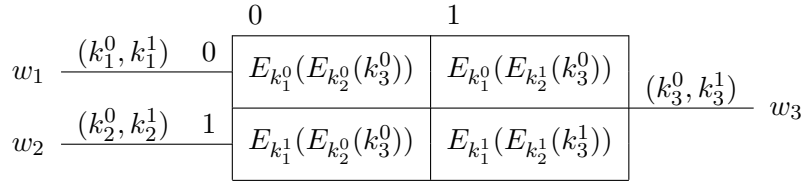
		b_2 = 0	b_2 = 1
b_1	0	0	0
	1	0	1

are the same or different.

4.3.1 General Idea of Yao Circuits

As already mentioned, the purpose of Yao circuits is to compute a function of several inputs in a way, that nobody learns more than the actual result and his own input.
The basic idea is that the a priori defined boolean circuit C, will be garbled by player P_1 by encrypting the values of C's truth table with randomly chosen keys. After that, P_1 runs an OT protocol with player P_2 to provide the decryption keys corresponding to P_2's input. With these keys, P_2 is able to decrypt one single entry in the truth table, representing the result of the computation.
The easiest way to illustrate this, is with an example: Let's assume there are two players, each having an input bit b_1 and b_2 and want to compute the binary operation AND ($\wedge$). The truth table for the AND operation is defined in Table 4.1. The circuit C computing the binary AND has two input wires w_1 and w_2 and one output wire w_3. Player P_1 randomly chooses two encryption keys, which map to binary 0 and binary 1, for each of the three wires w_i in C as (k_i^0, k_i^1). P_1 further replaces the values of the truth table with the corresponding output wire keys and encrypts these keys with the corresponding input wire keys as depicted in Figure 4.2. The resulting circuit C' is called the *garbled circuit*. Player P_1 sends the garbled truth table along with the key $k_1^{b_1}$ for the first input wire w_1, corresponding to his input bit b_1, and the mapping between the output keys (k_3^0, k_3^1) and the representing binary values. An OT protocol is executed between the two players, P_1 inputs the input wire keys (k_2^0, k_2^1) for the second input wire w_2 and P_2 inputs his input bit b_2. P_2 then has all the needed information to evaluate the garbled circuit C'. With the received input wire key $k_1^{b_1}$ and the obtained input wire key $k_2^{b_2}$ for the second wire from the OT, he is able to decrypt exactly one entry of the garbled truth table. With the output key bit mapping, P_2 can map the decrypted key $k_3^{b_1 \wedge b_2}$ back to the actual binary value, obtaining the result of the Yao computation.

Figure 4.2: Garbled circuit C' computing the binary relation AND with the corresponding truth table



4.3.2 Security Flaws in Yao Circuits and Countermeasures

The standard Yao protocol for secure function evaluation, as described above, is not secure against malicious adversaries. There are mainly three reasons for that, which are highlighted and handled by Lindell et al. in [3].

The first reason is, that the circuit constructor P_1 may send P_2 a garbled circuit that computes a completely different function than specified. The approach for solving this problem is to use the cut-and-choose technique. P_1 sends P_2 l garbled circuits C'_i and P_2 asks P_1 to open all but one of the circuits C'_b (chosen at random) in order to check that they are correctly constructed, and indeed compute the intended function.
With the parameter l, the notion of the deterrence factor ϵ comes again into play. For player P_1 to successfully cheat in the construction of the circuits, he has to predict b and is required to correctly construct all circuits except C'_b. There are l possible values for the choice b, and since b is randomly chosen with a uniform distribution over all possible values, the probability of correctly guessing it's value is $\frac{1}{l}$. The probability that P_1 will get caught in incorrectly constructing the circuits is at least $1 - \frac{1}{l}$, which is one part[3] of the deterrence factor ϵ of this subprotocol.

The second reason why the standard protocol is not secure against malicious adversaries, is because the used OT subprotocol within the Yao protocol is only secure in the presence of semi-honest adversaries. It is definitely not enough to just secure the Yao circuit computation without having an OT protocol, which is secure in the new adversary model. This is because a malicious adversary could cheat in the OT subprotocol, corrupting the whole Yao protocol, but without being detected. But this problem is easily solved by using the new OT protocol from Section 4.2.2, which is secure against covert adversaries. The security parameter k of the OT protocol needs to be set in a way, such that the deterrence factor is at least as high as the deterrence factor of the Yao protocol.

The third issue is a bit more tricky. In order to conform to the explicit-

[3]See Section 5.1.3 for the exact value of the deterrence factor of the final protocol

cheat definition of the covert adversary model, it must not be possible that the decision of the malicious party, whether to cheat or not, is made dependent on the honest parties' inputs or on the output. Consider a corrupted P_1 that behaves exactly like an honest P_1, except that in the oblivious transfers, it inputs an invalid key in the place of the key associated with 0 as the first bit of P_2. The result is that if the first bit of P_2's input is 1, then the protocol succeeds and no problem arises. However, if the first bit of P_2's input is 0, then the protocol will always fail and P_2 will always detect cheating. Thus, P_1's decision to cheat may depend on P_2's private input, something that is impossible in the explicit-cheat definition.
In order to make P_1's decision to cheat independent of P_2's input, the constructed circuit does not compute the function $f(x_1, x_2)$ directly but $g(x_1, x_2^1, \ldots, x_2^m) = f(x_1, \bigoplus_{i=1}^{m} x_2^i)$, where x_2 is split into m shares. So P_2 randomly chooses $x_2^1, \ldots, x_2^{m-1}$ and sets the last value $x_2^m = (\bigoplus_{i=1}^{m-1} x_2^i) \oplus x_2$. This modification does not change the result, but as long as P_1 does not provide invalid keys for all m shares, the probability of failure is independent of P_2's actual input.
The probability that P_2 catches P_1 cheating is $1 - 2^{-m+1}$, since $m - 1$ invalid shares are detected with exactly this probability. Beside the parameter l, this solution brings another security parameter m to the Yao protocol's deterrence factor definition.

The detailed description of the Yao's protocol steps can be found in Section B.3 of the appendix.

4.4 Mixnets

Mixnets are multi party protocols making it very hard to trace some sent data back to the original sender. Mixnets are a chain of proxy servers, each performing some anonymization operations over the data. In our case, a mixnet is performed over a matrix M, where the mixing operations are re-encryptions and a random permutation of the data elements in M.
The mixing step itself is done by first choosing and applying a random matrix permutation P, where the permuted matrix M' is transformed by $M' = P \cdot M \cdot P^{-1}$, which swaps rows and columns of M. Then, each element $M'(i, j)$ of the permuted matrix is randomized by adding an encryption of 0 to it, which does not change the encrypted value of the element due to the additively homomorphic encryption scheme used:

$$E_y(M'(i, j) + 0) = E_y(0) \cdot E_y(M'(i, j))$$

The matrix M contains the encrypted ME (number of missing events) and RK (rank of the event) values, which at the end are decrypted to identify all time-consecutive events. Therefore, it is needed in order to guarantee privacy, that no partner can deduce any sensitive information about the supply chain by looking at the broadcasted encrypted ME and RK values and

the decrypted matrix at the end.

4.4.1 Security Flaws in the Mixnet Protocol and Countermeasures

The mixnet protocol used in [1] only guarantees correctness and privacy in the semi-honest adversary model. As soon as covert adversaries are present, the mixnet has a critical flaw in terms of correctness.
The problem is that an adversary can replace every $E_y(ME(e_i, e_j))$ entry with $E_y(0)$ to make the resulting N_{ME} value always being a sum of 0. The N_{ME} value always being 0 means that in every check of missing events between each pair of observations, no events were missing, which further means that the players assumes a fully correct trace of the events and no clones will ever been detected.
A reliable countermeasure is, like in the previous subprotocols, the cut-and-choose technique. The idea is always the same. For the case of the mixnet, the mixing player creates h anonymized matrices M'_i and the other players decide the matrix M'_b, which is used for continuation of the protocol. For all other matrices $M'_i, i \neq b$ he has to reveal the permutation matrix P_i and the used randomness for the re-encryption of the elements, so that the others can verify the correctness of the mixing step.
As in all other subprotocols, for a corrupted player in order to successfully cheat, he needs to predict the right index b and must correctly perform the mixing for all other matrices. The probability of guessing a value which is uniformly chosen at random in the range $[1, \ldots, h]$ is exactly $\frac{1}{h}$. The probability that a cheating player is detected doing so is therefore $1 - \frac{1}{h}$, which is the deterrence factor ϵ of the subprotocol.

4.4.2 Secure Mixnet against Covert Adversaries

Unfortunately, there has no mixnet protocol already been presented, which guarantees security against covert adversaries, provides the needed functionality and is relatively simple and fast. For this reason, we developed a new mixnet which basically builds up on the one presented in [1] and uses the cut-and-choose technique to deal with covert adversaries.

Every party makes h different random permutations and for $h - 1$, he must prove to every other player that these are correct permutations. Then, the not chosen matrix is used by the next party to continue the mixing. After each broadcast of the mixed matrices in a round j, the players jointly compute an index b_j which indicates the matrix that is used for the continuation. For the first mixing step, every party P_i broadcasts his values. Then P_1 creates h permutations with randomizations of these values and

broadcasts the h matrices M_1^j, $j \in \{1, \ldots, h\}$ to every party. Then P_1 must prove, that M_1^j, $j \neq b_1$ are correct permutations of all the broadcasted values by revealing the used randomness for the randomization. So every player can check if the matrices M_1^j, $j \neq b_1$ are randomizations of the broadcasted values when using the revealed randomness. If every party has accepted, the matrix $M_1^{b_1}$ is used by the next player P_2 for the next mixing step. The protocol continues the same way for every following steps i, the matrices M_i^j, $j \in \{1, \ldots, h\}$ are broadcasted and the permutations and randomness for all $h-1$ matrices, except for the matrix $M_i^{b_i}$, are broadcasted. Every party can then check, if the matrices M_i^j, $j \neq b_1$ are correct permutations of the matrix $M_{i-1}^{b_{i-1}}$. If all parties accept the permutations, the matrix $M_i^{b_i}$ is passed to the next party P_{i+1}. This is done until all of the n parties executed its mixing step. The final matrix $M_n^{b_n}$ is then the result of the mixnet subprotocol and is used by the clone detection protocol for the next steps.

Index Computation Protocol

In each round i, after the broadcast of the l matrices, every player P_j locally define a number x_j and broadcasts a commitment $y_j = Com_{r_j}(x_j)$ to it. After everybody has received all the commitments, every player broadcasts the actual number x'_j and the used randomness for the commitment r'_j. Every player checks if the broadcasted r'_j indeed is the opening of the commitment y_j, namely $y_j = Com_{r_j}(x_j) \stackrel{?}{=} Com_{r'_j}(x'_j)$. The definition of the commitment function Com is known to every player. The players define a set V and put every player which broadcasted a valid opening of the commitment into this set. For the last step, everybody sums up the values of players belonging to the set V modulo h, resulting in the index which define the permutation used for the continuation of the mixnet protocol, $b_i = \sum_{P_j \in V} x_j \bmod h$.

The detailed step-by-step description of the new developed mixnet subprotocol can be found in Section B.4 of the appendix.

Deterrence Factor of the Mixnet Protocol

The computation of the deterrence factor ϵ is analogous to the previous presented subprotocols. Likewise in the OT and Yao scheme, a corrupted party needs to predict the index b in order to successfully cheat in the mixing phase. Since all numbers x_j are randomly chosen, the sum of them is random and uniformly distributed over all possible values too. For this reason, the probability of detecting a cheating player is therefore $1 - \frac{1}{h}$, and so is the deterrence factor ϵ of the mixnet.

4.5 Improved Privacy-preserving Protocol

In this section, we put everything together and describe the steps of our newly designed privacy-preserving clone detection protocol, which guarantees security in the covert adversary model.
Before the protocol can be executed between the partners of the supply chain, they obtain the necessary information to establish a connection between each of them. This information is obtained from a so called discovery service (DS), where partners store observation-related non-sensitive information like the ID and the address of their local databases. So upon a partner's request for clone detection on the supply chain for a given ID, the DS is used to bootstrap the network for the protocol execution.

The protocol steps are the following:

1. All players agree on the security parameters t for the threshold of the key generation, k, l, h for the cut-and-choose of the OT, Yao and mixnet respectively and m for the number of input sharings for the Yao circuits. A common choice to achieve a deterrence factor of $\epsilon = 0.5$ would be $k = h = 2$ and $l = m = 3$.

2. All players jointly generate a public key $y = (N, G)$ and a shared secret key x for an instance of the additively homomorphic Paillier's cryptosystem using the distributed key generation scheme as described above.

3. For all pairs of events $(e_i, e_j), i \neq j$, the players owing events e_i and e_j jointly evaluate $E_y(GT(e_i, e_j) := [e_i(T) > e_j(T)])$ running a Yao protocol with parameters l and m, and k for the OT. The result of the function is $E_y(1)$ and $E_y(0)$ mapped to the output wire representing 1 ($e_i(T) > e_j(T)$) and 0 ($e_i(T) < e_j(T)$) respectively. After the circuit evaluation, the players run the protocol again but with switched roles, where the sender becomes the receiver and vice versa to evaluate $E_y(GT(e_j, e_i))$.

4. For all pairs of events $(e_i, e_j), i \neq j$, the players owning events e_i and e_j jointly evaluate $E_y(ME(e_i, e_j))$, which corresponds to the number of missing events between these two events e_i and e_j, using a Yao circuit. The outputs of the three-output Yao circuit are $E_y(0)$ mapping to 0 and $E_y(1), E_y(2)$ and $E_y(3)$ mapping to 1 for the first, second and third output wire respectively. $E_y(i), i \in [0, 3]$ represents i missing events between the considered two events. The player owning e_i opens the outputs of the circuit, and sums them up to get an encryption of $ME(e_i, e_j)$. The summation works due to the additive homomorphic characteristic of the encryption, $E_y(\sum_{k=1}^{3} N_{ME,k}) = \prod_{k=1}^{3} E_y(N_{ME,k})$.

5. For each of his events e_i, each player adds the bits he got in step 3, obtaining the encryption under y of the rank of the event e_i in a time-sorted list of all events. Each player then performs this addition locally by calculating $E_y(RK(e_i) = E_y(\sum_{j \neq i} GT(e_i, e_j)) = \prod_{j \neq i} E_y(GT(e_i, e_j))$. The received values from all players can be arranged in a quadratic matrix M which looks like:

$$M = \begin{pmatrix} E_y(RK(e_1)) & \dots & E_y(ME(e_1, e_{N_{TE}})) \\ \vdots & \ddots & \vdots \\ E_y(ME(e_{N_{TE}}, e_1)) & \dots & E_y(RK(e_{N_{TE}})) \end{pmatrix}$$

   Each player contributes a row in the matrix and all the diagonal elements $M_{i,i}$ are the ranks of the event e_i, whereas all other elements $M_{i,j}$ corresponds to the number of missing events between events e_i and e_j. N_{TE} is the total number of tag events owned by all the n players.

6. The players perform a mixnet over M. Each player P_i randomly permutes the matrix and randomizes each element. The final matrix is denoted as M_n.

7. The players collaboratively decrypt the diagonal of M_n. After that two time-consecutive events can be identified and therefore the total number of missing events can be computed by summing up the entries for all identified time-consecutive event pairs. (As an example: event e_i has rank a, $M_n(i, i) = a$, and event e_j has rank $a+1$, $M_n(j, j) = a+1$, then the corresponding number of missing events between these two events is at $M_n(i, j)$.) If all identified numbers of missing events are determined and summed up, the result is an encryption under y of the total number of missing events N_{ME}.

8. The players collaboratively decrypt the computed value $E_y(N_{ME})$. Each player can now evaluate based on its significance level α and misread probability p_{mr} the presence of clones.

CHAPTER 5

Evaluation

In this chapter, we evaluate our new developed protocol according to three different metrics. First of all, the security analysis is done for all subprotocol and for the final protocol. We point out the possible attacking points of a covert adversaries in these protocols and the countermeasures which defeat such attacks.
Further, we discuss the computational and time complexity of the protocols. As a next step we briefly recapitulate the detection probabilities of the subprotocols and make some statements about the overall detection probability of the final protocol in correlation with the individual deterrence factors.
In the last section, we present the results of our simulations in order to evaluate the performance of the protocols. We describe how the tests are set up, reason about the obtained results and compare them to the semi-honest results.

5.1 Security and Complexity Analysis

Before we move to the performance evaluation with the diagrams, we present the theoretical part of the evaluation for the security and complexity metrics. First, we analyze our four subprotocols and then we put everything together and evaluate the final composed protocol.

5.1.1 DKG Analysis

Our distributed key generation protocol, as described in Section 4.1, is secure against covert adversaries. We used Shamir's secret sharing scheme [15]

where the secret key is shared in a *t-out-of-m* manner. With this sharing mechanism, it is not possible anymore, for a malicious adversary to reconstruct the secret key alone, in order to read the ciphertexts in plain. The threshold t needs to be set according to the assumed environment and in proportion to the number of players in the protocol. If less than the majority of the players are assumed to be malicious, the value for t can be chosen as $\frac{n}{2}$, where n is the total number of players.

DKG Complexity

Unlike for the other subprotocols, the complexity does not really depend on the security parameter t in the DKG protocol. In fact, the complexity is more or less constant, aside from the number of players, which marginally influences the time and bandwidth complexity for obvious reasons like the computation and distribution of the shares.
The threshold t has an influence on the size of the polynomial, which increases the computation time a little bit. Altough the number of raised coefficients increases when t gets bigger, the resulting influence on the sent data sizes is very small, since the other values in the messages do not depend on t.
The key generation step at the beginning and the chosen key sizes have the biggest weights in the complexity definition.
Beside the creation of the shares, the proof construction and verification operations in step 2 and 3, only slightly increase the computational complexity compared to the Fouque and Stern [16] version.

5.1.2 OT Analysis

As already described in Section 4.2, our 1-out-of-2 oblivious transfer protocol guarantees security against covert adversaries. The key point is, like for all other subprotocols, the use of the cut-and-choose technique. Here we want to give a brief idea, how to argue about the two security related criteria, the correctness and the privacy. Since a covert adversary may actively corrupt the protocol in order to get some sensitive informations, we show what an adversary may want to do and explain why this is not possible, or at least only with a small probability.

Concerning the *correctness*, one may think that the sender S can input whatever he wants, so in step 6 he can basically compute everything (by using another c_0 and c_1) and pass this to R. But this is not a security flaw, since in the ideal model he can do the same. But if you assume that only correct encrypted values will be decrypted to a value different from $\perp$, the situation described above would not happen.

On the other hand, *privacy* must be guaranteed in both directions. Note that as long as a cheating party does not deviate from the protocol, privacy is guaranteed, since a corrupted receiver would only learn a single value x_0 or x_1 and a corrupted sender can not infer σ from the unopened ciphertexts due to the semantically secure encryption scheme.
But since we deal with covert adversaries, these assumptions are no longer true. So for the other direction of the privacy, meaning to corrupt the sender's privacy, the receiver R may input twice an encryption of the value 1 for the chosen key pair in step 3. However the sending party catches him cheating with a probability of $1 - \frac{1}{k}$ in the encryption-generation challenge (EGC) in step 4.
The key generation in step 1 of the OT protocol has a cheat detection mechanism too. A corrupted receiver has to open the used randomness in the key generation for all except the chosen key pair. Since there are k key pairs generated and the sender chooses b uniformly at random, the adversarial receiver is caught with probability $1 - \frac{1}{k}$ in the key-generation challenge (KGC) even after incorrectly generating one single key pair.

This leads us to the deterrence factor ϵ of the OT subprotocol which is directly defined by the choice of the security parameter k. Theoretically, it is clear that larger values of k bring a higher detection probability ϵ, since the deterrence factor is computed as $\epsilon = 1 - \frac{1}{k}$. Intuitively spoken, the higher the parameter k, the larger is the interval b is chosen from and the guessing probability of the receiver is directly correlated with this interval size.
In theory, the value k can be chosen from the interval $[2, \infty)$, since at least two possibilities are needed to obtain an ϵ-value strictly greater than 0 and the higher k is, the closer is the detection probability to 1.
In practice, there are boundaries for k of course, because a higher deterrence factor comes with a prize in the time and memory complexity.

OT Complexity

The complexity of the OT protocol heavily depends on the parameter k. Obviously, the memory consumption and runtime of the protocol is linearly dependent of the chosen k. The protocol generates k key pairs and locally stores the used randomness for each pair. In step 1 and 2, the receiver sends the key pairs and the random coins over the network to the sender, which increases the communication complexity with the same factor. For the operations in step 3 and 4, the same happens with the encrypted bits and the opening. The rest of the protocol then does not depend on k anymore and therefore has always the same complexity.
Dependent of the used encryption scheme, the key generation in step 1 can be a very time-consuming operation. As a solution, one can define a sufficiently

large set of key pairs in advance, so that within the actual execution, the protocol can just randomly choose from this set, instead of newly generating them again on every protocol run.

5.1.3 Yao Analysis

Since we developed a clone detection protocol which deals with covert adversaries, the Yao circuit computation guarantees security in this model too. As already described in Section 4.3, there are two security parameters within the Yao subprotocol that defines the deterrence factor ϵ, the number of input shares m and the cut-and-choose value l.
Since both techniques, the cut-and-choose and the input sharing, influence the security, the deterrence factor definition contains both probability computations resulting in $\epsilon = (1 - \frac{1}{l})(1 - 2^{-m+1})$.

Beside the already mentioned attacking possibilities for covert adversaries, like corrupting the circuit construction or making the decision to cheat dependent of the other player's input bits, there are some more security related attacks for a corrupted party in terms of correctness and privacy.

Concerning the *correctness*, player P_1 could input to the OT twice the tuple for just one bit: $([k^1_{w_{n+i},0}, \ldots, k^l_{w_{n+i},0}], [k^1_{w_{n+i},0}, \ldots, k^l_{w_{n+i},0}])$ to fix the input of the other player P_2. But this succeeds only with probability $\frac{1}{l}$, since in point 3 of step 9, P_2 checks that the received keys in the i-th OT equals the keys $k^j_{w_{n+i},z_2^i}$ from the opening.
Further, for some computations it is needed that P_1 at the end gets the result as well. But it can be, that P_2 doesn't send him the result (not a problem since not sending is seen as cheating) or the wrong one. But for this case P_1 and P_2 compute the circuit once again but with the roles switched, meaning that P_2 acts as P_1 and vice versa.

The *privacy* has to be guaranteed again for both players.
For the first player P_1, his commitments $\{c^j_{w_i,0}, c^j_{w_i,1}\}$ are sent in random order, so P_2 doesn't know which key belongs to 0 and 1. P_1 just sends the decommitments to the keys for his input bits and since they are in random order, P_2 doesn't know to which bit it belongs.
For the other direction, P_2's privacy is guaranteed by the privacy guarantee of the OT protocol (due to the semantically secure encryption scheme) as described in the previous section.

Unlike all other subprotocols, the Yao protocol has two security parameters. This gives the ability to play around with the setting of l and m, in order to achieve a certain detection probability ϵ. Obviously, there are often more than one configuration of the two parameters for a given ϵ. Most of

the time there is only one single configuration which is optimal in terms of time and bandwidth consumption. Other settings beside the optimal choice have often a large difference in the complexity. In Section 5.2, we analyze and present the impact of the different settings in our implementation.
Here we just want to give the explanation for these differences by looking at the complexity of the Yao subprotocol.

Yao Complexity

Certainly, the complexity of Yao's circuit computation algorithm against covert adversaries depends on both parameters l and m. In order to find out, which one influences the complexity in which way, we have to understand what they denote.
The defined circuit C' in step 1 has $n + mn$ input wires and internally consists of $(m-1) \cdot n$ XOR circuits, to reassemble the input from the input shares, together with one GT or BT circuit, depending on the circuit's function. So the size of the circuit C' is linearly dependent of the parameter m, which infects the communication complexity and the memory complexity as well.
Further, both players run $m \cdot n$ executions of an OT protocol as stated in step 4. Since we need to run an OT for every one of P_2's input bits, a larger number m leads to a blowup of the number of OT runs needed.
However, a homomorphic encryption step in the OT is needed for every of the $m \cdot n$ key pairs per circuit, the total number of these encryption steps is therefore $m \cdot n \cdot l$. So as a first very important observation, we can see that the time complexity for the involved OT depends on this product. In order to optimize this time consumption, we have to find the optimal choice of the two parameters (l_{opt}, m_{opt}) such that no other pair (l_i, m_i), giving the same deterrence factor, satisfies the inequality $m_i \cdot n \cdot l_i < m_{opt} \cdot n \cdot l_{opt}$.
For the bandwidth consumption of the Yao protocol, the criteria is the same. The size of the garbled circuits and the number of random coins which need to be sent within the protocol execution, linearly depend on these three numbers. So if you are able to set l and m such that the product $l \cdot m$ remains small, you can keep the data which is sent over the network small too.

There are cases where two settings for l and m, for a certain ϵ, are the opposite pair of each other like (x, y) and (y, x). For such pairs, the product of $l \cdot m$ is always the same, resulting in the same complexity for the OT. This doesn't mean that both configurations are optimal though. This can easily be seen in Figure 5.5, where the two pairs $(l = 2, m = 30)$ and $(l = 30, m = 2)$ approximately give the same ϵ, but where the pair $(l = 2, m = 30)$ has the better time and bandwidth consumption.
A possible and feasible reason for this fact is the construction of the circuits. As already stated, the two circuits computing the GT and ME function-

ality, are a composition of several XOR circuits with a GT or ME circuit respectively. The XOR circuits can be implemented very simple by just one gate. The GT circuit, which takes the input of the first player and the output of the XOR chains, consists of 32 smaller circuits arranged in a consecutive chain. These are needed to deal with 32 bit inputs instead of just one single bit. The ME circuit is even slightly larger than the GT circuit. The circuit, which is located after the XOR chains, itself consists of a GT circuit together with another small circuit.
It is obvious, that the size of a GT or ME circuit is much larger than the size of a simple XOR circuit. Likewise, the construction time for building such a circuit, needs therefore more computational power and is more time-consuming compared to a XOR circuit.
As a consequence, whenever there are two settings (x, y) and (y, x) giving the same ϵ, the pair with the smaller l value generally gives the better time and bandwidth consumption results.
A simple example shows this effect: The pairs $(3, 2)$ and $(2, 3)$ give both a deterrence factor of $\epsilon = \frac{1}{3}$. Since the product $l \cdot m = 6$ is the same for both, the last criteria discussed must be applied. Indeed the pair $(2, 3)$ gives better results in terms of time and bandwidth consumption as it can be seen in Figure 5.5. The reason is that the $(2, 3)$ circuit has 2 GT and 4 XOR circuits, whereas the $(3, 2)$ circuit has 3 GT and 3 XOR circuits. However the latter circuit has one XOR gate less, the additional GT circuit overwhelms the saving of this XOR gate.

5.1.4 Mixnet Analysis

Our new developed mixnet protocol employs the cut-and-choose technique in order to provide security in the covert adversary model. As already discussed in Section 4.4, a covert adversary can attack the protocol in a way which is not possible for semi-honest adversaries.
We want to recapture the security attacks trying to corrupt the correctness and privacy definitions of a covert adversary.

Concerning the *correctness* of the protocol, a malicious adversary could just replace every ME-value in the matrix with $E_y(0)$ resulting that the protocol will not detect any clones anymore. But since every mixing player must prove to every other player, that the resulting matrix is a correct permutation of the input matrix, such an attack will be detected with a probability at least $1 - \frac{1}{h}$.
Since the basic mixing is not different than in the protocol for semi-honest adversaries, the correctness follows from the fact, that every player permutes and randomizes the chosen matrix.

The *privacy* follows from the encryption scheme, where decryption can

only be done if the adversary controls more than t players.

Since the mixnet protocol just has one security parameter h, the deterrence factor ϵ is directly correlated with h. A malicious party trying to corrupt the correctness as described above, succeeds in exactly one case. He has to predict the right index e_i from step 2 of the protocol description and must correctly perform the permutation and randomization for all other matrices $M_i^j, j \neq e_i$. Since the index e_i is chosen uniformly at random, as shown below, the success probability for the adversary is $\frac{1}{h}$. Therefore, the deterrence factor of the mixnet subprotocol is the complement of the success probability $1 - \frac{1}{h}$.

Uniform Random Index Computation

In order to define the deterrence factor computation as above, it is required that the choice of the index e_i in round i is uniformly distributed in the interval $[1, h]$ and random.
We disclaim here for a formal proof and rather give the idea and the argumentation. The first observation is that the value e_i, defined as the sum of some other values x_j, $e_i = \sum_{j=0}^{n} x_j \mod h$, is random even if just one x_k is truly random and the other values are let's say 1. So if a corrupted mixing player wants to cheat, he must control all other players as well, in order to fix the choice e_i. Otherwise, if there is even one single honest party among the other players, the choice e_i will be random and only predictable by the cheater with the probability $\frac{1}{h}$. But the case where all players are corrupted is nonsense, since we want to protect honest players from cheaters, and there are no honest parties involved anymore.
The uniform distribution of e_i follows from the same fact as the randomness. Since a random value x_k is not predictable by the cheaters and e_i is computed as the sum modulo h, their choice indeed influence the result, but they have no clue in which direction. So the distribution of the index e_i cannot be influenced, as long as at least one honest party is involved in the computation.

Mixnet Complexity

Like in all protocols using the cut-and-choose technique, the complexity is closely connected with the security parameter. Another, probably yet more important factor, is the number of players involved in the protocol and the number of observations.
In every mixing round of the protocol, the permuting player generates h mixed matrices and broadcasts them over the network. As a response, the other players first broadcast a commitment of their values and the opening after that. Then the mixing player reveals the randomness used for the per-

mutations and randomizations for the not chosen index. As a result, the time consumption for generating the mixing and the bandwidth usage for the broadcasts, grows linearly with the number of h.

The most influencing factor is the number of players. As this number increases, not only the bandwidth consumption increases due to the broadcasts. The time consumption increases on one hand, due to the increased complexity in coordinating the communication between all players. On the other hand, the increased time usage is caused by the additional mixing rounds in the protocol, since every participating party needs to perform a mixing operation.

The last factor is the number of observations, which determine the dimension of the matrix. Normally, for real world application scenarios, each player has exactly two observations per ID, except the first and the last partner in the chain having just one observation. So the number of observations is generally determined by the number of players in the protocol. But for theoretical reasons, we run simulations with different numbers of observations. From these results, one can see that this parameter has a big effect on the time and especially on the bandwidth consumption. As long as the observation count increases, the size of the matrix increases too, which further means that more randomization operations have to be performed and the permutation gets larger as well. These reasons effects the time consumption.
The different dimensions of the matrices influences the bandwidth complexity in the same way. As soon as the matrices grow in size, the communication costs increase, since now larger matrices and openings of the mixing need to be broadcasted. The problem is, that the size of the matrix is not linear in the number of observations, but quadratic. This is because for every new observation o_{new}, the value of the rank function $RK(o_{new})$ and the missing events function $ME(o_{new}, o_i)$ and $ME(o_i, o_{new})$ between all already existing observations o_i must be contained in the matrix. This fact illustrates the quadratic size of the matrix in the number of observations.

5.1.5 Protocol Analysis

In this section, we put all subprotocols together for the concluding analysis of the final privacy-preserving clone detection protocol against covert adversaries.
The protocol guarantees security in the covert adversary model, since all subprotocols involved provide security for this definition.
The central question is, how to set the mentioned security parameters of the subprotocols in order to achieve a certain overall deterrence factor ϵ. The goal is to set them as low as needed, in order to have an optimal time and

bandwidth consumption, for a given value of ϵ.
The answer to this question is actually pretty easy, after thinking about the notion of security for the final protocol. A very brief and informal definition of security including the notion of the detection probability, is that a cheater cannot corrupt the correctness or privacy aspects at any point in the protocol, without being caught with a probability less than ϵ.
It directly follows from this definition, that it is required that the deterrence factor of every subprotocol is at least ϵ_o, in order to achieve an overall detection probability of ϵ_o. Otherwise an attacker can corrupt the protocol at some point of the subprotocol providing a smaller deterrence factor ϵ_s and therefore gets caught cheating only with this smaller probability, which is not permitted in the security definition.

The complexity of the final protocol is basically somehow the sum of all subprotocols. The most carrying weight subprotocol is certainly the Yao, because for each pair of observations, one GT and one ME function has to be evaluated by a Yao circuit. The DKG has to be executed just one time at the beginning of the protocol, and it's time consumption is relatively low anyway. The mixnet subprotocol has a similar contribution to the overall complexity, since it is executed only once too, but with a much higher weight though. Likewise, it is executed only once after all Yao computations at the end of the protocol. The OT is entirely embedded in the Yao subprotocol, but is executed multiple times within one Yao run. So the OT subprotocol is obviously the bottleneck of the whole protocol. If it is not possible to implement the OT primitive in a efficient way, its complexity will dominate the entire protocol implementation.

5.2 Performance Evaluation

After the theoretical analysis about security and complexity, we present and evaluate our protocol implementation in practice. A rough description of the architecture and design of the implementation with some statements about the application flow, can be found in Chapter 6.
We performed several simulations on the subprotocols, in order to show the impact of the different security parameter settings, as well as on the final clone detection protocol.
In Section 5.2.6, we compare our results to the ones obtained from the recently presented implementation against semi-honest adversaries [1].
The interesting measurements are the running time and the communication complexity in terms of the size of the messages sent over the network during the protocol.

5.2.1 Simulation Environment

We run our simulations on a local network of 10 identical computers, each one having a Pentium4 3 GHz dual core CPU with 1 GB RAM installed and are connected through a switched 10/100 Mbs LAN. All machines run on Linux with the Fedora Core release 7 distribution with identical hardware configurations.
The tools used to collect the data are the *dumpcap* utility, that comes with Wireshark [25] for analyzing the network traffic, and the UNIX *time* command for measuring the running time of the protocols. These two measurements are the average values from 10 simulation runs and the bandwidth results are the computed average of the sent message sizes per player.

5.2.2 OT Simulations

The simulation results for the oblivious transfer subprotocol are depicted in Figure 5.1. The diagram shows the linearly increasing time and bandwidth consumption as the number of challenges increases.
The setup of the simulation, was to compute an OT as it is in the Yao GT function computation with parameters $l = m = 3$. That means that one player inputs $n \cdot l \cdot m$ keys of size 80 bits and the other player gets the keys corresponding to his σ values.
In our implementation we achieved results between 60 s and 300 s for the running time, and from 0.3 MB to 0.9 MB in terms of sent message sizes.
In Figure 5.2, we run some simulations with a modified version of our OT protocol. This implementation provides the basic notion of a single-bit 1-out-of-2 OT, where the sender inputs two 80-bit keys (wk_0, wk_1) and the receiver has a single choice bit σ. After the execution the receiver gets the key wk_σ.
In the final protocol, a batch version of the OT implementation was used. That means, the OT protocol takes as input from the sender all the $l \cdot m \cdot n$ wire keys and the receiver inputs a sequence of choice bits of the same length. Our OT protocol provides the same functionality as when sequentially executing the single-bit-OT protocol $l \cdot m \cdot n = 288$ times, for the optimal choice of $l = m = 3$. However the time and bandwidth consumption results in Figure 5.1 are by far not $l \cdot m \cdot n$ times larger than for one single-bit-OT. One reason for this is that the time to build up the communication channels, is done just once in the batch version, whereas it would be counted 288 times for the composed single version. Another reason is the communication overhead itself, where the number of messages used is again 288 times more than in the batch version where just one message of every OT-message-type is used. So in general it can be said, that the computational and bandwidth complexity in terms of the encryption, the checking operations and the decommitments, is the same for the batch version and the

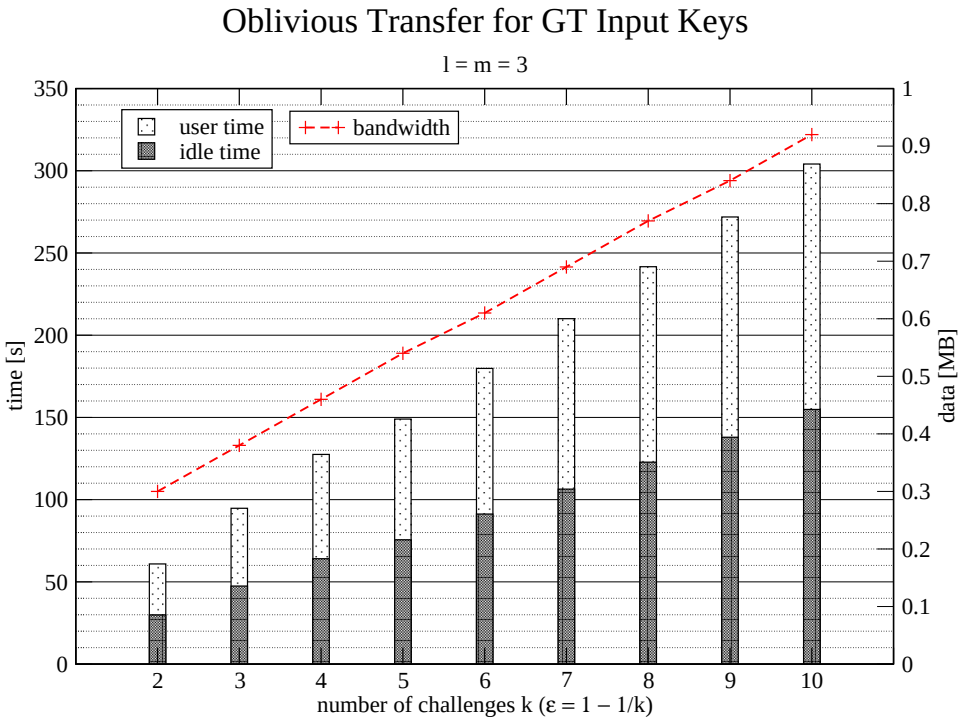


Figure 5.1: Running time and bandwidth consumption of an OT protocol run as in a Yao GT circuit computation with parameters $l = m = 3$

$l \cdot m \cdot n$ times composed single OT implementation. The only issue affecting the running time, is the multiple times executed step 1 of the protocol, where the public keys for the encryption are constructed. Since we use the Paillier encryption scheme for this, and since the generation of public keys is a very time-consuming operation, the multiple executions of this step take much more time than to just execute it once, namely 288 times more.
So the only difference in complexity comes with the described overhead of the channel setup, the number of messages and the generation of the public key pairs.

5.2.3 Yao Simulations

Since there are two security parameters to adjust the deterrence factor of the Yao subprotocol, we can run different interesting simulations. As it can be seen in Figure 5.3 and Figure 5.4, we can fix one parameter and vary the other and vice versa, in order to show the impact on the deterrence factor, the time and the communication complexity. It can easily be seen, that the two diagrams look very similar. The reason for this is that the two curves for the ϵ value are almost the same, which further means, that the two settings for a certain ϵ are the complement pairs of each other. So for example the

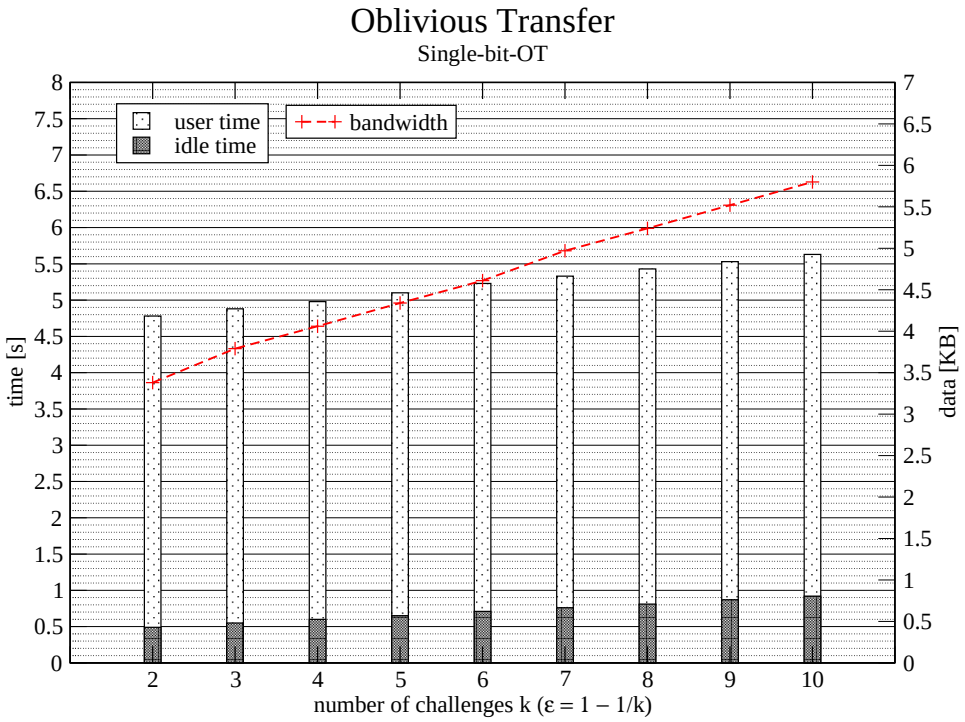


Figure 5.2: Running time and bandwidth consumption of an OT protocol run with 2 input keys of 80 bits and one single selection bit σ

time and bandwidth values for the point 6 on the x-axis denotes the (l, m)-pair $(3, 6)$ in Figure 5.3 and $(6, 3)$ in Figure 5.4. Both settings provide a deterrence factor slightly higher than 0.6. As explained in Section 5.1.3, the expected results should not be far away from each other. Indeed, this is the case here, where the time difference is less than 10 seconds ($\approx 6\%$) and the difference in bandwidth consumption is about 0.1 MB ($\approx 10\%$).
Another simulation which demonstrates this situation even better is shown in Figure 5.5. We evaluated different settings for l and m which achieve a deterrence factor which is around 0.5. The first plot in the diagram is the optimal choice $l = m = 3$ for $\epsilon = 0.5$ which has a running time of 70 seconds and a bandwidth consumption of about 0.5 MB. The remaining points can be seen as 2 similar groups, where particularly the settings giving a deterrence factor between 0.45 and 0.5 are interesting. However the ϵ-values are not exactly the same for the two mountain shaped point groups, but the described effect can be seen very good. The group where l has the value 2, outperforms the other group with $m = 2$ in every simulation and for both metrics. Even though the first group gives the higher ϵ-values than the corresponding settings in the other group, the time and bandwidth usage is smaller for the described reasons.

Another described relation between the two security parameters and the complexity in Section 5.1.3, can be seen by looking at the results for the two settings $l = m = 3$ and $l = 2, m = 10$. Both configurations provide a detection probability of $\epsilon = 0.5$. But as it can be seen with a short computation, the optimal case has the factor $l \cdot m = 9$ for the complexity, whereas the other has the factor $l \cdot m = 20$. Therefore the time and bandwidth consumption for the latter case is expected to be about twice as much as for the optimal case. In fact, the running time of 137 seconds is closely twice the time of the optimal case which is about 70 seconds. The computation for the bandwidth shows exactly the same result. The used 0.9 MB there, is approximately twice the data volume of 0.47 MB for the optimal setting. The running time of 70 seconds for a Yao GT computation with $\epsilon = 0.5$ is just by 10 seconds larger than the result for the OT run involved. This is feasible, since the most time-consuming part of our Yao implementation is the oblivious transfer. The remaining 10 seconds are used for the circuit construction and mainly for the verification in step 9. The bandwidth consumption of this setting can be explained in the same way. The OT subprotocol needs 0.3 MB as message sizes, whereas the Yao needs about 0.5 MB for the same detection probability. The remaining 0.2 MB are used to send the garbled circuits and the openings in step 8. For the other settings, the bandwidth needed to send the circuits and the decommitments is larger than for the $\epsilon = 0.5$ case, but that is correct, since with higher values of l and m, the circuits and decommitments get bigger. The same is true for the running time for higher detection probabilities, since it takes more time to construct a larger circuit and to verify more gates in the opening.
The last simulation is to show the dependence between the two metrics and the different choices of the deterrence factor. In Figure 5.6, we show the results for the optimal settings for ϵ between 0.25 and 0.8. On the x-axis below the deterrence factor, the optimal setting of the Yao parameters are shown and the minimal required OT parameter k, which gives at least the given ϵ. It can be seen, that the values for both metrics grow in a quadratic manner as the detection probability increases. A reason for this, is the fact that not only the Yao parameters have to be increased in order to achieve higher probabilities, but the cut-and-choose parameter of the OT as well. So with higher values for l and m, the number of OT within the Yao computation increases too. But the running time increases more than linearly, since the time consumption of the OT itself increases.
Another simulation could be to use less bits to represent the timestamp or the location of an observation. So if we consider $n = 16$ instead of $n = 32$, we can save resources for obvious reasons. Since the time and bandwidth complexity depends only on the three values l, m and n, we could reduce the running time as well as the bandwidth by a factor of 2, if we are able to represent timestamps and locations with half of the bits.

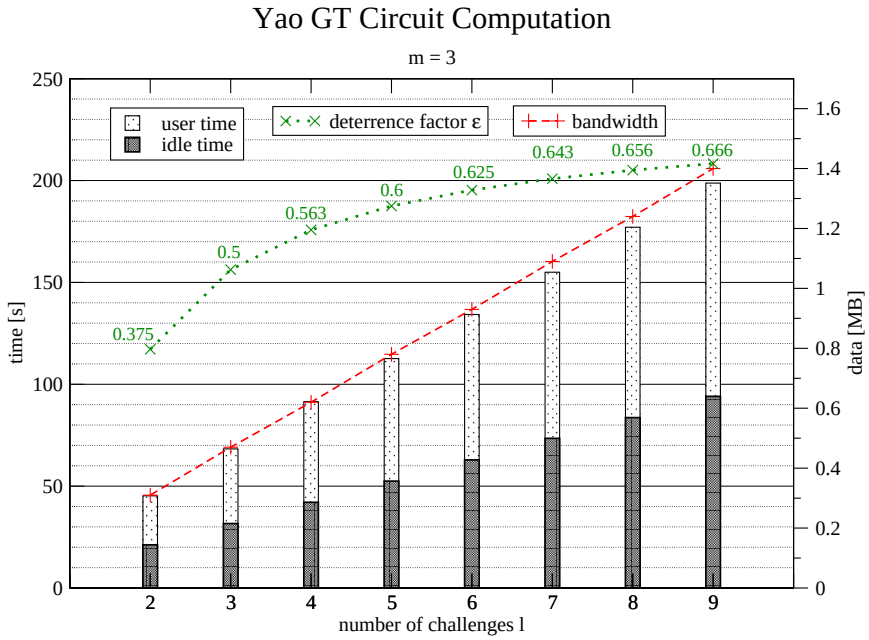


Figure 5.3: Running time and bandwidth consumption of a Yao protocol run computing the GT function, depending on the parameter l with fixed $m = 3$

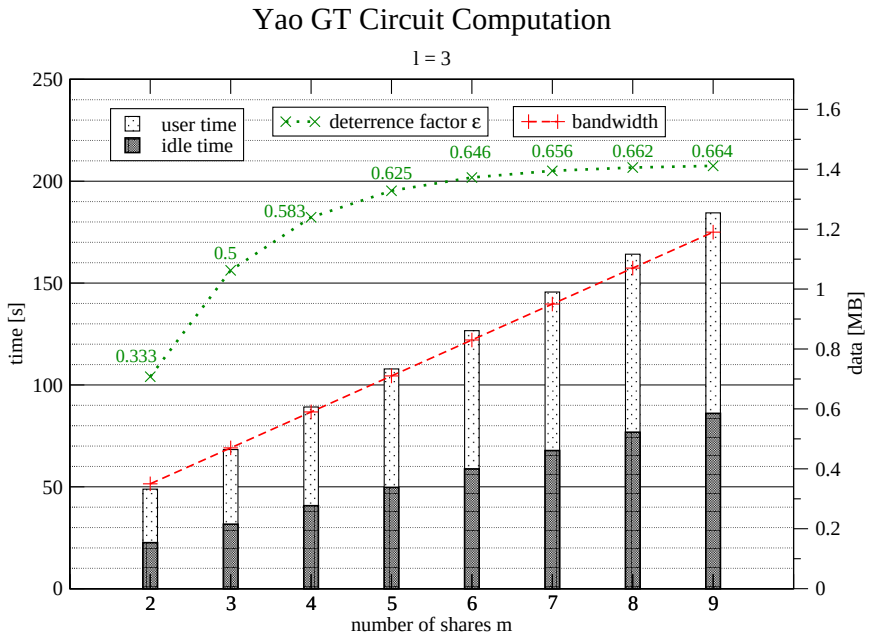


Figure 5.4: Running time and bandwidth consumption of a Yao protocol run computing the GT function, depending on the parameter m with fixed $l = 3$

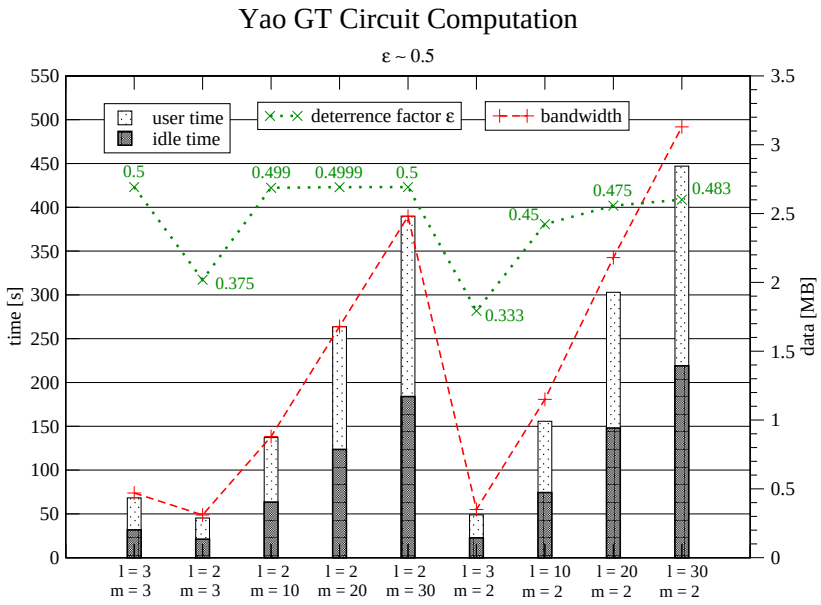


Figure 5.5: Running time and bandwidth consumption of a Yao protocol run computing the GT function, depending on the parameters (l, m) for different ϵ-values around 0.5

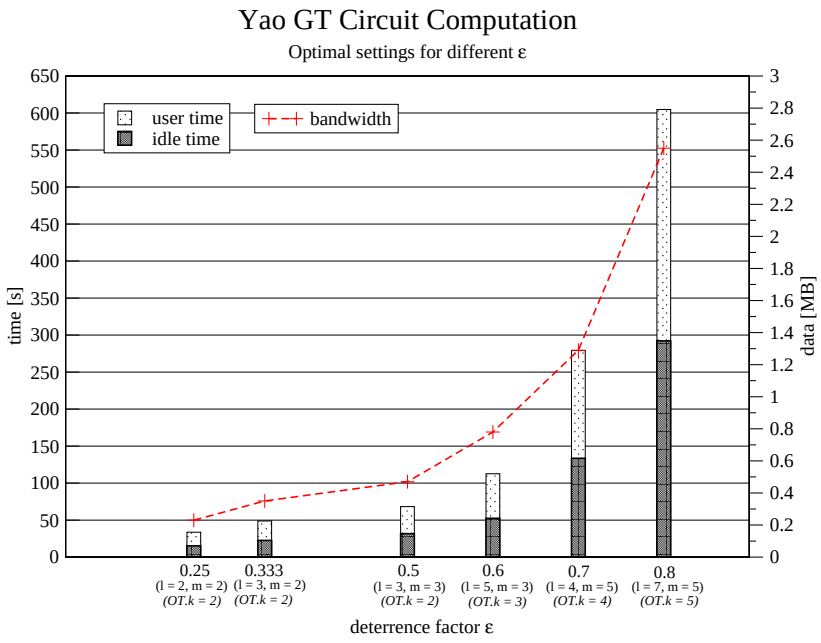


Figure 5.6: Running time and bandwidth consumption of a Yao protocol run computing the GT function, with the optimal (l, m) settings for different ϵ-values

5.2.4 Mixnet Simulations

As already seen in Section 5.1.4, the mixnet subprotocol has 3 factors influencing the measurements. In our tests, we run simulations for different number of players and different number of observations. Within each simulation, we evaluated the protocol for all possible values of the cut-and-choose parameter h between 2 and 10.
In all four test situations, the configuration of h has a linear impact on the time and bandwidth consumption of the subprotocol. A more interesting effect shows up, as soon as one of the other two parameters change. When comparing Figure 5.7 with the results in Figure 5.10, in order to analyze the impact of varying the number of players while keeping the same number of observations, it can be observed, that the size of the messages sent does not differ much. This is because the number of observations is relatively small, and therefore has not such a big effect on the message size. The bigger effect can be seen between 5 players and 10 players where the number of observations is 18 in both cases (Figure 5.8 and 5.9).
The time difference between two cases where just the number of players change shows another very nice result of our implementation. As expected and described in Section 5.1.4, the running time with 10 players is about twice the running time of the 5 players' setting. Indeed, this can be seen as a result in the two corresponding diagrams for 10 observations. The effect for 18 observations is very similar, but the result for 10 players is slightly more than factor 2 larger than the one for 5 players. This is feasible as well, since with more observations, the messages get larger and therefore the transmission time increases too.
The difference between the two real world scenario simulations with 5 players (8 observations) and 10 players (18 observations), is very big, especially for the time metric. Both weighty factors described, the number of players and the number of observations, are increased in these two situations which explain this effect.

5.2.5 Protocol Simulations

For the simulation of the final protocol, we defined the deterrence factor to be $\epsilon = 0.5$ and run the whole protocol with different number of players. As it can be seen in Figure 5.11, we did the measurements for the number of players between 3 and 10. As expected, the running time and bandwidth usage linearly grows while increasing the number of players. Although the total number of Yao computations increases quadratically whenever a new player with two new observations is added to the protocol, the additional computation for each player is at most 4, which is linear. This is due to the fact, that each player has his own machine and therefore the workload is

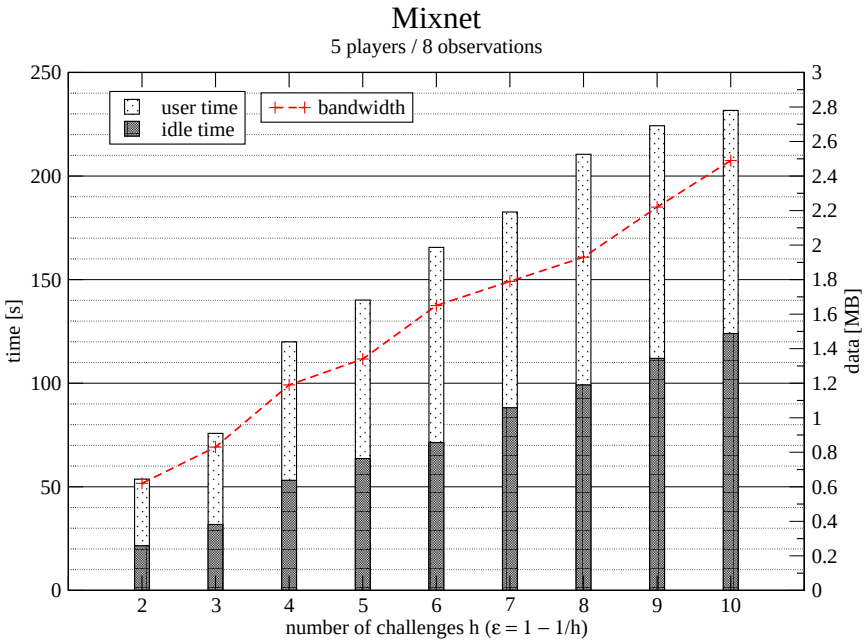


Figure 5.7: Running time and bandwidth consumption of a real world mixnet protocol run with 5 players and 8 observations for different ϵ-values

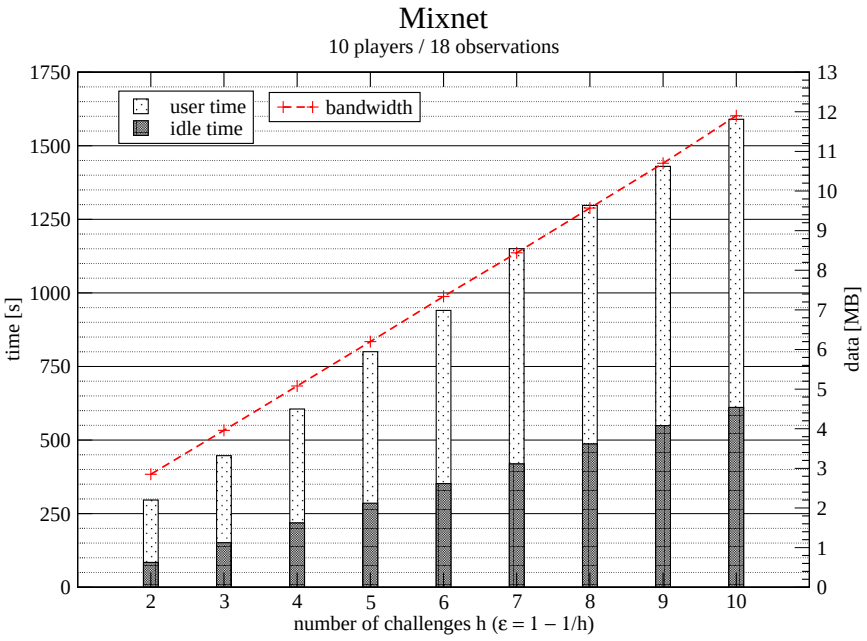


Figure 5.8: Running time and bandwidth consumption of a real world mixnet protocol run with 10 players and 18 observations for different ϵ-values

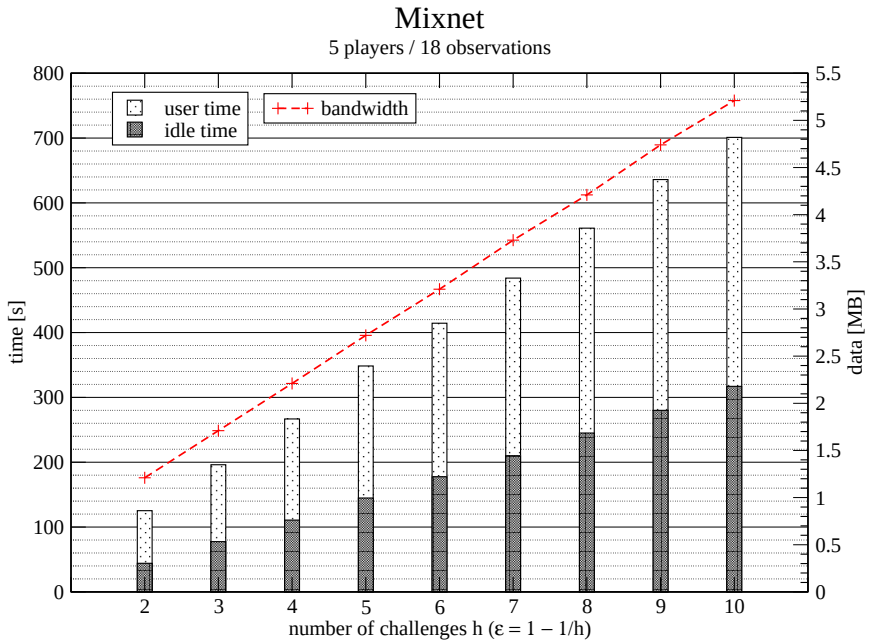


Figure 5.9: Running time and bandwidth consumption of a mixnet protocol run with 5 players and 18 observations for different ϵ-values

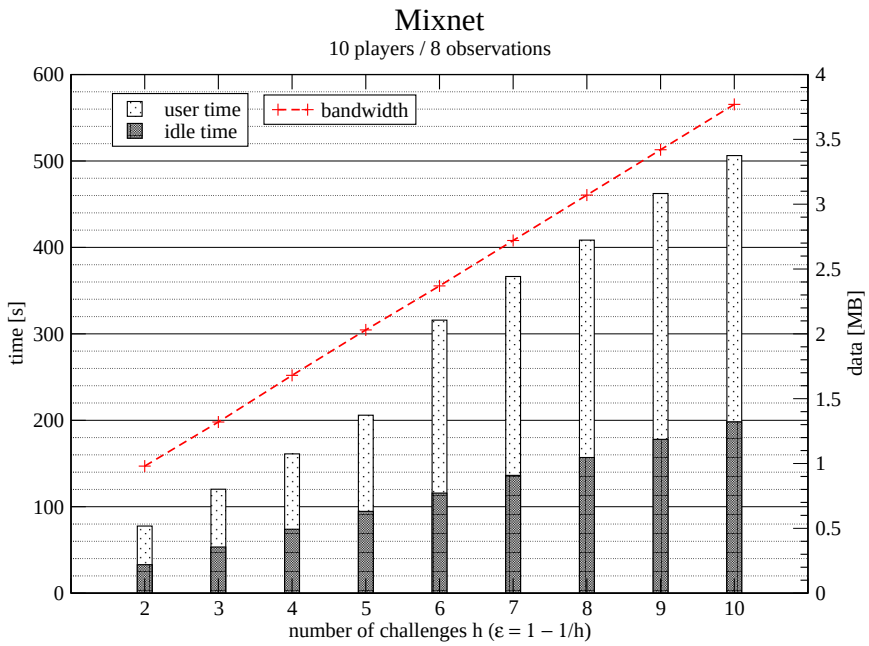


Figure 5.10: Running time and bandwidth consumption of a mixnet protocol run with 10 players and 8 observations for different ϵ-values

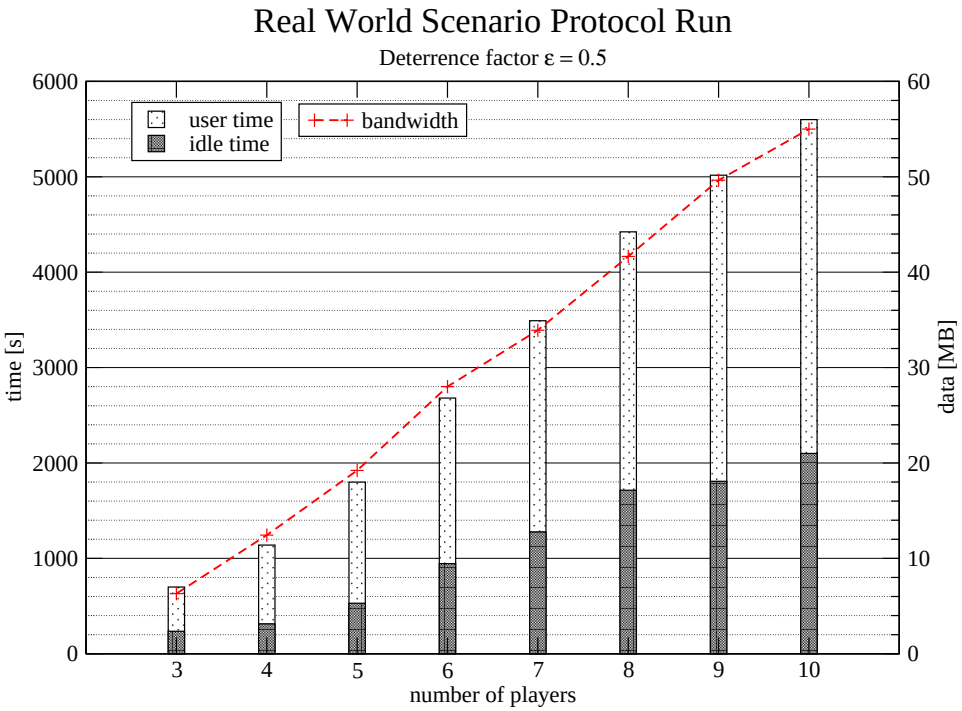


Figure 5.11: Running time and bandwidth consumption of the final protocol run with deterrence factor $\epsilon = 0.5$ for different numbers of players

distributed over all participating nodes.
The execution time for 5 players is about 1800 seconds. With 5 players, there are 8 observations, so in average each player has to compute 12 GT and 12 ME circuits. From the diagram in Figure 5.5 for the optimal setting $l = m = 3$, it can be seen that one Yao circuit computation takes about 70 seconds. So for the Yao, each player needs in average $24 \cdot 70s = 1680s$, which is approximately 93% of the total time. The mixnet subprotocol with 5 players and 8 observations takes 60 seconds as shown in Figure 5.7. Summing up both values gives a time consumption of the Yao and mixnet of 1740 seconds. The remaining 60 seconds are needed by the DKG subprotocol which includes the actual key generation and the collaborative decryption at the end of the protocol, which indeed is feasible.
However, the number of bits n for timestamps and locations only affects the Yao computation, the effect of considering smaller values affects the final protocol as well. Since the Yao is by far the most time and bandwidth consuming subprotocol, an assumed value of $n = 16$ would almost reduce the total running time and sent message sizes by a factor of 2.

5.2.6 Result Comparison

In this section, we try to compare our results with Fellmann's results in [1]. Since his protocol guarantees security in the semi-honest adversary model, we cannot compare the actual implementations. But what we can do, is to estimate the costs to change the protocol's security from the semi-honest to the covert adversary definition. Unfortunately, there are no detailed results for any subprotocol presented in Fellmann's work. In order to not just compare the two implementations on the final protocol level, we extracted the Yao subprotocol and run it with a GT circuit computation for two 32 bit timestamps as input. For the time and bandwidth consumption of the extracted Yao subprotocol, we measured about 7 seconds for the running time and approximately 35 KB for the message sizes. The comparison shows, that our Yao implementation for $\epsilon = 0.5$, takes about 10 times the running time of the semi-honest protocol and has almost 14 times larger message sizes per player. But these results are not so far out, since in the $l = m = 3$ case, the number of OT runs is 9 times the number of OT runs in the semi-honest implementation. The bandwidth usage is more than 9 times larger, because not only the additional data due to the cut-and-choose technique contributes to this result, but as well the additional XOR gates in the circuits due to the input sharing.
Like in our simulations, he measured the total running time of the final protocol and the average bandwidth usage per player. His results goes from about 20 seconds for 3 players up to 170 seconds for 10 players in term of the running time and between 0.4 MB and 3.5 MB for the bandwidth usage. As it can be seen, the running time of our implementation is about 35 times slower than the semi-honest protocol. For the bandwidth, the sent message sizes per player are about 15 times larger. Of course, it is obvious, that an implementation providing security in a stronger model, has a larger complexity. The question is, whether it is worth to use the covert adversary protocol instead of the semi-honest one, although the differences in time and bandwidth consumption are relatively big. This probably depends on the application of the protocol and how the business requirements to the detection mechanism are specified.
The big differences in time are once due to the additional cryptographic complexity like commitments, and due to the employed cut-and-choose technique, which obviously multiplies the time and data complexity.
The cut-and-choose technique certainly has an effect on the bandwidth consumption too, but the additional cryptographic primitives do not contribute that much to the message sizes. That is as well a reason, that the difference in the bandwidth usage is by far not as big as the difference of the running time.
However, our implementation is definitely not fully optimized, but the mentioned key ideas are valid and the trend of the results is visible. In order to

present more detailed explanations for the differences, the implementation of our protocol needs to be better optimized first.

CHAPTER 6

Implementation

In this chapter, we briefly describe the implementation of our privacy-preserving clone detection protocol against covert adversaries.
The code is completely written in Java using the NIO (New I/O) library for the connection management and communication between the partners. The protocol has the ability to be ran locally on just one machine where each player is simulated by a separate thread, or rather for practical usage with the remote mode, where every player is on his own host. The communication between all nodes is implemented with the Java's SocketChannels class and is message based.
A participating partner in the protocol is represented with the NetworkPlayer class, which extends the Thread class. A player also owns a ProtocolState object, where the actual protocol logic and message handling is implemented. A NetworkPlayer basically blocks in a while loop, and gets woken up on every message arriving on a socket. Then, the message is deserialized and passed to the ProtocolState object, which checks what kind of message it is and handles it accordingly.
The ProtocolState class is in fact just a big if-then-else block, to determine the message type, and has a big number of handler methods which handles the messages according to their types and continues the protocol by sending the next message.

There are more than 20 different types of messages in the whole protocol. Every subprotocol has its own bunch of message types. For simplicity of the message handling and for the reason that often every step of a subprotocol needs to transfer different sets of information, each step of a subprotocol has

its own message type.
In the next few sections we shortly mention the most important parts of the implementation and describe in a few words, how they work.

6.1 Messaging Module

One of the central parts of the implementation is the messaging module. It consists of a connection manager, which keeps track of the connections to the other nodes, and the messager itself which is responsible for the whole message communication.
The messager mainly consists of a sending part as a separate thread, implemented in the SenderThread class, and a receiving part implemented in the Messager class.

6.1.1 Sending Messages

The SenderThread class is a separate thread responsible for sending the messages from the protocol to the intended partner nodes. It has a internal list of messages which are waiting to be sent. Once the protocol wants to send a message over the network, the message is not immediately sent, but added to this list first.
As long as the protocol has not finished, the sending thread is in a loop waiting to send messages. After adding a message to the list, the sending thread gets woken up by the Java's *notify()* instruction. Then it will send all messages currently waiting in the list and gives control back to the protocol execution thread by the Java's *wait()* method after that.

6.1.2 Receiving Messages

The Messager class is responsible for reading messages from the socket channels. As soon as the NetworkPlayer thread detects a message receiving on the socket channel, it tries to read the message with the messager object. The read method then starts reading out the receiving buffer of the socket channel. If it is not able to read the whole message object, it puts the NetworkPlayer thread to sleep for a short while and gives control to the sending thread.
This is an important mechanism because otherwise it may happen that two players want to send each other messages and run into a deadlock. This would be possible due to the blocking of the sending and the relatively large message sizes compared to the fixed Java's receive buffer size[1]. So the situation would be the following, that both players are sending their messages until the receive buffer of the other party is full. Then no one could proceed

[1]The Java's receive buffer has the fixed size of 130 KB, whereas the protocol messages can have sizes up to almost 600 KB

in sending the rest of the message and since the sending wouldn't be completed, the players couldn't start reading and clearing the buffer.
But with the separate sender thread, the sending could be made non-blocking, meaning that whenever it recognize that the sending can not proceed, the sending thread is put to sleep for a short while and the NetworkPlayer's thread gets the control and can start reading out the own receive buffer. With this approach a deadlock like in the described case cannot occur.

6.2 Protocol Logic

Most of the protocol logic related code is in the ProtocolState class. Every time a message is read from the socket channel, the messager passes it to the ProtocolState object, which basically determines the type of the message, performs the needed computation and sends on the next message specified. This object has also a huge number of fields for storing the current state of the protocol. Most of these fields are Java HashMaps, mapping between the index of the Yao circuit and the corresponding data, like commitments or generated keys.

6.3 DKG

The distributed key generation algorithm is implemented as described in Section 4.1. The code is entirely situated within the ProtocolState class. The implementation of the DKG subprotocol needs two different message types. One message contains the public key of the Paillier instance from step 1 as described in Section B.1 of the appendix, which is broadcasted at the beginning of the DKG. The other message contains the raised coefficients $A_{i,k}$, the raised shares $y_{i,j}$, the encrypted shares $Y_{i,j}$ and the proofs for the PVE.
The key size of the encryption scheme is configured to 512 bits.

6.4 OT

The oblivious transfer subprotocol is like the DKG part of the ProtocolState class. The OT implementation has five different message types and follows exactly the description presented in Section 4.2. For the first four steps described in Section B.2, one type of message exists per step. The last message type is used for step 6 of the protocol to send the two computed ciphertexts $\tilde{c}_0$ and $\tilde{c}_1$ to the receiving party.

6.5 Yao

The Yao implementation is probably the most complicated and biggest part of the whole protocol. It is implemented according to the description in Section 4.3 and especially following the steps as in Section B.3.
The subprotocol needs six special message types aside from the message types used for the OT. As for the subprotocols before, the Yao implementation is part of the ProtocolState class.
In order to increase the performance and reduce the used time, the start of the Yao computations are threaded. For each computation which needs to be done, a YaoKickOffThread is spawned, preparing the inputs of the receiving party by splitting them into m shares. After that, the thread sends a message to the other player to express that he can start the actual Yao protocol. The thread is then closed and the ProtocolState takes control.
For the other party, on receiving such a message, a YaoResponseThread is spawned which basically starts with step 3 of the protocol description. The purpose of this thread is to speed up the whole Yao computation of all event pairs, like the counterpart thread from before. These two threads achieve the improvements in terms of time consumption by parallelizing the splitting and key generation steps.
The chosen key size for the input wire keys in our implementation is 80 bits.

6.6 Mixnet

The mixnet subprotocol is implemented according to the description in Section 4.4 with the detailed steps from Section B.4. The implementation uses 5 different message types for the communication. Two messages are used for the index computation denoting the matrix used for the continuation of the mixing steps. One message is needed at the beginning of the protocol to collect all the lines to build the matrix M from all the players. The remaining two messages are used to open the permutations and randomizations for the not chosen matrices $M_i^j, j \neq e'_i$ in round i.

CHAPTER 7

Related Work

Secure multi party computation (SMC) protocols for covert adversaries are studied for the first time in the paper of Aumann and Lindell [3]. They introduced the definition of covert adversaries in their work and argued about the motivation for this new model. They presented two protocols for the oblivious transfer and Yao circuit computation. In [7], Goyal et al. went further in the development of secure multi party computation primitives in the covert adversary model by improving the computational and time complexity of Aumann and Lindell's solution. Another work in the same field was the paper of Damgård, Geisler and Nielsen [20], showing how to compile a passively secure protocol for honest majority into one that is secure against covert attacks.
Secure multi party computation is also covered in a slight different way in the paper of Malkhi et al. [21]. They introduce a full-fledged system called Fairplay [22] where the boolean circuit is represented in a own language called SHDL and evaluated in the manner suggested by Yao in [10].
Actual implementations of such protocols against covert adversaries are not presented by now. A Java implementation of Yao circuits for the GT and ME operations are used in [1]. However this is the only implementation for these two operations since generic constructions like SMC frameworks are often too slow for special functions and need to be newly designed for every operation.
But there are also solutions [5, 6], which allows the computation of any polynomial-time function in a secure multi party manner and achieve security in the active adversary model. But as already mentioned, these protocols are not feasible in practice due to the large computational complexity.

A privacy-preserving clone detection mechanism for RFID-enabled supply chains was presented in [1]. There are more papers about trace-based clone detection solutions in [23] and [24]. They try to cope with incomplete traces caused by possible tag misreads and partners not sharing their tag observations. Their approaches are less practical for the dynamic structure of supply chains. All these proposed solutions however assume either a central repository that stores the partners' tag observations data or that they share the sensitive data in plaintext. In order to guarantee privacy, such mechanisms are not applicable to our scenario.

CHAPTER 8

Conclusion

We presented an improved privacy-preserving clone detection protocol for RFID-enabled supply chains. Our mechanism builds up on a recently presented semi-honest secure protocol, but guarantees security in terms of privacy and correctness against covert adversaries. With the used secure multi party computation primitives, the supply chain partners are able to execute the protocol among each other without revealing any sensitive information, even in the case when the adversaries' capabilities move from passive to active attacks.
The general approach to achieve security in the covert adversary model for the used subprotocols, is to use the cut-and-choose technique.
We first presented the four new developed subprotocols (DKG, OT, Yao, Mixnet) and explained their intended purpose and the interaction within the final protocol. After that, we implemented the subprotocols as well as the final protocol, in order to run simulations and get informations about the performance. In the next chapter, we analyzed the security and evaluated the performance of our protocols. We derived the impact to the performance of the different settings for the deterrence factor ϵ and reasoned about optimal choices for the security parameters and the security guarantee of the composed protocol.
As a result, we were able to show that the presented protocol is feasible for practical applications. Although the implementation is not fully optimized yet, we managed to successfully execute a real world protocol run among 10 partners in a much faster way than known protocols which guarantee security in the malicious adversary model.

8.1 Future Work

The bottleneck in the protocol's implementation are the encryption and decryption operations in the oblivious transfer and Yao subprotocols. So future work may address the time improvement of the encryption operations. A possible approach could be the use of elliptic curve cryptography (ECC) for which it is believed that the same level of security afforded by an RSA-based scheme with large key sizes can be achieved with a much smaller elliptic curve group. This allows to reduce storage and processing time of the intense time-consuming encryption and decryption operations.

Appendices

APPENDIX A

Definitions

A.1 Covert Adversary Definitions

There are three model definitions for covert adversaries (from weaker to stronger):

Failed-simulation Formulation: It is guaranteed, that if the output distributions of the ideal-model simulator and the real protocol execution can be distinguished with some probability Δ, then the honest parties will detect cheating by a corrupted party with probability at least $\epsilon \cdot \Delta$, where ϵ is the deterrence factor.

Explicit-cheat Formulation: The ideal model is modified, so that a special cheat instruction can be sent by the adversary to the trusted party. Upon receiving such an instruction, the trusted party hands all the honest parties' inputs to the adversary. Then, it tosses coins and with probability ϵ announces to the honest parties that cheating has taken place (by sending the message corrupted$_i$ where party P_i is the corrupted party that sent the cheat instruction).

Strong Explicit-cheat Formulation: It is like the above explicit-cheat formulation, except that the adversary only receives the honest parties' inputs in the case that the honest parties do not detect its cheating.

A.2 Paillier's Cryptosystem

Paillier's cryptosystem [17] is a probabilistic asymmetric encryption scheme for public key cryptography. The security is based on the decisional composite residuosity assumption[1]. Let p and q be two large RSA moduli randomly chosen, such that $gcd(pq, (p-1)(q-1)) = 1$. Set $N = pq$ and $\lambda = lcm(p-1, q-1)$. Select a random integer $G \in \mathbb{Z}^*_{N^2}$ whose order is a large multiple of N modulo N^2. This can be done by checking the existence of the modular multiplicative inverse $\mu = (L(g^\lambda \mod N^2))^{-1} \mod N$, where the function L is defined as $L(x) = \frac{x-1}{n}$. Let g be of order a prime q in $\mathbb{Z}^*_p$. The private key for decryption is (λ, μ) and the public key for encryption is (N, G).
A message $M \in \mathbb{Z}_N$ is encrypted as $c = G^M u^N \mod N^2$, where $u \in_R \mathbb{Z}^*_N$. The scheme is additive homomorphic, means that given only the public-key and the encryption of messages M_1 and M_2, one can compute the encryption of $M_1 + M_2$.
The decryption of a ciphertext c is computed as $M = \frac{L(c^\lambda \mod N^2)}{G^\lambda \mod N^2)} \mod N$.

[1] See [17] for the definition and the security analysis

APPENDIX B

Subprotocols in Detail

B.1 DKG Protocol in Detail

The protocol works as follows:

1. In the initialization stage, each player performs the key generation algorithm of Paillier's cryptosystem. For $i = 1$ to l, the public keys $PK_i = (G_i, N_i)$ are published and the server P_i secretly stores SK_i. The value N_i is a RSA modulus, G_i is an element in $\mathbb{Z}^*_{N_i^2}$ of order a multiple of N_i and $SK_i = \lambda(N_i)$.
   The values p, q, g are global defined values for all players.

2. P_i generates a random $s_{i,0}$, sets $a_{i,0} = s_{i,0}$ and chooses $a_{i,k}$ at random from $\mathbb{Z}_q$ for $1 \leq k \leq t$. The numbers $a_{i,0}, \ldots, a_{i,t}$ define the polynomial $f_i(X) = \sum_{k=0}^{t} a_{i,k} X^k \in \mathbb{Z}_q[X]$. Then, he computes $s_{i,j} = f_i(j) \mod q$. He broadcasts : for $k = 0, \ldots, t$, $A_{i,k} = g^{a_{i,k}} \mod p$ and $y_{i,j} = g^{s_{i,j}} \mod p$, $Y_{i,j} = G_j^{s_{i,j}} u_{i,j}^{N_i} \mod N_j^2$, and a proof $(e_{i,j}, w_{i,j}, z_{i,j})$. *(See below for detailed values of the proof.)*

3. Then, for each $1 \leq i, j \leq l$, the servers verify that : $\prod_{k=0}^{t} A_{i,k}^{j^k} = \prod_{k=0}^{t} g^{a_{i,k} j^k} = g^{\sum_{k=0}^{t} a_{i,k} j^k} = g^{f_i(j)} \mod p$ and check whether $g^{f_i(j)} \mod p$ is equal to $y_{i,j}$ in order to verify that the distribution is correct. The servers also verify the proofs $(e_{i,j}, w_{i,j}, z_{i,j})$ and if $y_{i,j}^q = 1 \mod p$ for $1 \leq i, j \leq l$. *(See below for detailed verification of the proof.)*

4. The set Q of qualified servers is defined from the players who have correctly played. The others are disqualified.

5. Player P_j decrypts $Y_{i,j}$ and obtains $s_{i,j}$ for $1 \leq i \leq l$. He stores the parts $s_{i,j}$ for $i \in Q$ and computes the public key as $\prod_{i \in Q} A_{i,0} = g^{f(0)} \mod p$ if we note $f(X) = \sum_{i \in Q} f_i(X)$. The share of the key obtained by player P_j is equal to $\sum_{i \in Q} s_{i,j} = f(j) \mod q$. The secret key s is shared in polynomial form with $f(j) \mod q$ and in additive form with $x_j \mod q$ between all players belonging to the set Q.

Broadcast of Proofs The following elements are sent as the proofs for the Publicly Verifiable Encryption (PVE):

- $e_{i,j} = H(g, G_j, y_{i,j}, Y_{i,j}, g^{r_{i,j}} \mod p, G_j^{r_{i,j}} b_{i,j}^{N_j} \mod N_j^2)$, where $r_{i,j} \in [0, \mathcal{M}), b_{i,j} \in \mathbb{Z}_{N_j}^*, \mathcal{M}$: message space.
- $w_{i,j} = b_{i,j} u_{i,j}^{e_{i,j}} \mod N_j$
- $z_{i,j} = r_{i,j} + e_{i,j} s_{i,j}$

Verification of Proofs The needed checks for the verification are the following:

- $e_{i,j} \stackrel{?}{=} H(g, G_j, y_{i,j}, Y_{i,j}, g^{z_{i,j}} y_{i,j}^{-e_{i,j}} \mod p, G_j^{z_{i,j}} w_{i,j}^{N_j} Y_{i,j}^{-e_{i,j}} \mod N_j^2)$
- $z_{i,j} \stackrel{?}{\in} [0, \mathcal{M})$
- $y_{i,j}^q \stackrel{?}{=} 1 \mod p$

B.2 OT Protocol in Detail

The protocol works as follows:

1. Receiver R generates $k = \text{poly}(n)$ sets of two pairs of keys $(pk_1^0, pk_2^0), ..., (pk_1^{k-1}, pk_2^{k-1})$ with random coins $r_G^{0,1}, r_G^{0,2}, \ldots, r_G^{k-1,1}, r_G^{k-1,2}$, where $r_G^{i,j}$ is used to generate key pair pk_j^i. R sends all the k tuples of public keys to the sender S.
2. *Key-generation challenge:* S chooses a random value $b \in [0, k-1]$ and sends it to R. R sends all the random-coins $r_G^{i,j}, i \in [0, k-1], j \in \{1, 2\}$ except $r_G^{b,1}$ and $r_G^{b,2}$ to S and S checks that the public keys for all received coins $r_G^{i,j}, i \neq b$ are the same that he has received from R in step 1. Let $pk_1 = pk_1^b$ and $pk_2 = pk_2^b$.
3. R chooses k random bits $\alpha_1, \alpha_2, \ldots, \alpha_k$ and computes encryptions of them where $\{\alpha_1, \ldots, \alpha_k\}$ are encrypted with new random coins $r'_{i,1}, i \in [1, k]$ and public key pk_1 and $\{1-\alpha_1, \ldots, 1-\alpha_k\}$ are encrypted with new random coins $r'_{i,0}, i \in [1, k]$ and public key pk_2 and sends the ciphertext

pairs $(c_1^0, c_1^1), (c_2^0, c_2^1), \ldots, (c_k^0, c_k^1)$, where (c_i^0, c_i^1) is the encryption of the pair $(1 - \alpha_i, \alpha_i)$, to S.

4. *Encryption-generation challenge:* S chooses a random value $b' \in [0, k]$ and sends it to R. R sends the random-coins $r'_{i,j}, i \in [1, k], j \in \{0, 1\}$ except $r'_{b',j}, j \in \{0, 1\}$ to S and S checks for every ciphertext pair (c_i^0, c_i^1), that one item is an encryption of 0 and the other an encryption of 1, if the random coins $r'_{i,0}$ and $r'_{i,1}$ are used.

5. R sends a "reordering" of the ciphertexts $\{c_{b'}^0, c_{b'}^1\}$ according to the value of σ. This means, that if $\sigma = 0$, c_0 is the ciphertext encrypting the value 1 and c_1 is the ciphertext encrypting the value 0. Otherwise if $\sigma = 1$ the mapping is vice versa. (This can be sent along with the openings in step 4.)

6. S uses the homomorphic property and the two ciphertexts c_0 and c_1 to compute $\tilde{c}_0 = x_0 \cdot_E c_0$ and $\tilde{c}_1 = x_1 \cdot_E c_1$ and sends them to R. The $\cdot_E$ operation is relative to the key pk_1 or pk_2 depending if c_i is an encryption under pk_1 or pk_2. (Note that one of the ciphertext is encrypted with pk_1 and the other is encrypted with pk_2.)

7. If now $\sigma = 0$, then R outputs $x_0 = D_{sk_1}(\tilde{c}_0)$ or $x_0 = D_{sk_2}(\tilde{c}_0)$ depending whether $\tilde{c}_0$ is encrypted under pk_1 or pk_2. Otherwise, if $\sigma = 1$, R decrypts $\tilde{c}_1$ and outputs the result.

8. If at any stage during the protocol, a party P does not receive the next message or an invalid message, then it outputs abort_P (unless it was already instructed to output corrupted_P).

B.3 Yao Protocol in Detail

The protocol works as follows:

1. The two parties define a new circuit C' that receives $m + 1$ inputs $x_1, x_2^1, \ldots, x_2^m$ each of length n and computes the function

$$f(x_1, \bigoplus_{i=1}^{m} x_2^i)$$

(Note that C' has $n+mn$ input wires which are denoted by $w_1, \ldots, w_n$ for x_1 and $w_{n+(i-1)m+1}, \ldots, w_{n+im}$ for x_2.)

2. P_2 splits his x_2 into m shares $x_2^1, \ldots, x_2^m$ of length n. These x_2^i are now P_2's new input z_2.

3. P_1 chooses l encryption keys for each $i = 1, \ldots, mn$ and $\beta = 0, 1$. The j-th key associated with a given i and β is denoted $k^j_{w_{n+i},\beta}$ denoting the key associated with the bit β for the input wire w_{n+i} in the j-th circuit.

4. P_1 and P_2 run mn executions of an oblivious transfer protocol as follows. In the i-th execution party P_1 inputs the pair
$$([k^1_{w_{n+i},0}, \ldots, k^l_{w_{n+i},0}], [k^1_{w_{n+i},1}, \ldots, k^l_{w_{n+i},1}])$$
and party P_2 inputs the bit z_2^i and receives the keys
$$[k^1_{w_{n+i},z_2^i}, \ldots, k^l_{w_{n+i},z_2^i}]$$

5. P_1 constructs l garbled circuits $GC_1, \ldots, GC_l$ of the circuit C'. The keys for the input wires $w_{n+1}, \ldots, w_{n+mn}$ in the garbled circuits are taken from above (i.e. in GC_j the keys associated with w_{n+1} are $k^j_{w_{n+i},0}$ and $k^j_{w_{n+i},1}$). The keys for the input wires $w_1, \ldots, w_n$ are chosen randomly, and are denoted in the same way. P_1 sends the l garbled circuits to P_2.

6. P_1 commits to the keys associated with its inputs and sends them as l vectors of pairs (one for each circuit); in the j-th vector the i-th pair is $\{c^j_{w_i,0}, c^j_{w_i,1}\}$ in a random order.

7. P_2 chooses a random index $\gamma \in_R \{1, \ldots, l\}$ and sends γ to P_1.

8. P_1 sends P_2 all of the keys for the inputs wires in all garbled circuits except for GC_γ together with the associated mappings and the decommitment values.

9. P_2 checks that everything that it received is in order, i.e. *(i)* the keys received for all input wires in circuits GC_j indeed decrypt the circuits and the decrypted circuits are all C', *(ii)* the decommitment values correctly open all the commitments that were received, and *(iii)* these decommitments reveal the keys $k^j_{w_j,\beta}$ that were sent for P_1's wires.

10. P_1 sends decommitments to the input keys associated with its input for the unopened circuit GC_γ.

11. P_2 checks that the values received are valid decommitments to the commitments received above. If not, it outputs abort_1 and if yes, it uses the keys to compute $C'(x_1, x_2) = C'(x_1, x_2^1, \ldots, x_2^m) = C(x_1, x_2)$, and outputs the result.

B.4 Mixnet Protocol in Detail

The protocol works as follows:

1. Every P_i broadcasts his $RK(e_i)$ and $ME(e_i, e_j)$ values to every other player. Every player P_j locally constructs the matrix M_1. The player P_1 generates h mixes of the matrix M_1 as $M_1^1, \ldots, M_1^h$ and broadcasts them to all other players.

2. The players jointly compute the first index b_1 and P_1 broadcasts all the used randomness for the mixing of $M_1^j, j \in [h], j \neq b_1$.

3. All other players check if the matrices $M_1^j, j \in [h], j \neq b_1$ are correct mixes of the matrix M_1 using the received random values. If not more than t players complain, the protocol continues with P_2 doing the same steps as P_1 before now with the matrix $M_2 = M_1^{b_1}$. Otherwise the protocol stops and declares P_1 as cheater.

4. If the protocol goes through all n players without stopping, the matrix $M_n^{b_n}$ is the resulting matrix of the protocol.

Bibliography

[1] L. Fellmann, "Privacy-preserving anti-counterfeiting for RFID-enhanced supply chains", Master thesis, ETH Zurich, 2009.

[2] D. Zanetti, L. Fellmann, and S. Čapkun, "Privacy-preserving Clone Detection for RFID-enabled Supply Chains", Department of Computer Science, ETH Zurich, Tech. Rep., 2009.

[3] Y. Aumann and Y. Lindell, "Security against covert adversaries: Efficient protocols for realistic adversaries", in *TCC*, 2007, pp. 137–156.

[4] The EPCglobal Network, The organization to achieve worldwide adoption and standardization of Electronic Product Code (EPC) technology, `http://www.epcglobalinc.org`

[5] O. Goldreich, *Basic Applications*, Foundations of Cryptography, vol. 2 (Cambridge University Press, Cambridge, 2004).

[6] O. Goldreich, S. Micali, A. Wigderson, How to play any mental game — a completeness theorem for protocols with honest majority, in 19*th STOC* (1987), pp. 218-229.

[7] V. Goyal, P. Mohassel and A. Smith. Efficient Two- and Multi-party Computation Protocols for Covert Adversaries. In Advances in Cryptology — *EUROCRYPT*, April 2008.

[8] M. Ben-Or, S. Goldwasser, A. Wigderson, Completeness theorems for non-cryptographic fault-tolerant distributed computation, in 20th *STOC* (1988), pp. 1–10

[9] D. Chaum, C. Crépeau, I. Damgard, Multi-party unconditionally secure protocols, in 20th *STOC* (1988), pp. 11–19

[10] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *SFCS* '86: Proceedings of the 27th Annual Symposium on Foundations of Computer Science, pages 162–167, Washington, DC, USA, 1986. IEEE Computer Society.

[11] Y. Desmedt. Society and group oriented cryptography: A new concept. In *CRYPTO'87*, volume 293 of LNCS, pages 120–127. Springer-Verlag, 1987.

[12] Y. Desmedt and Y. Frankel. Threshold cryptosystems. In *CRYPTO'89*, volume 435 of LNCS, pages 307–315. Springer-Verlag, 1990.

[13] T.P. Pedersen. A threshold cryptosystem without a trusted party. In *EUROCRYPT'91*, volume 547 of LNCS, pages 522–526. Springer-Verlag, 1991.

[14] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Distributed key generation for discrete-log based cryptosystems. In *EUROCRYPT'99*, volume 1592 of LNCS, pages 295–310. Springer-Verlag, 1999.

[15] Adi Shamir. How to share a secret. Commun. ACM, 22(11):612–613, 1979.

[16] P. Fouque and J. Stern. One round threshold discrete-log key generation without private channels. In *PKC'01*, volume 1992 of LNCS, pages 300-316. Springer-Verlag, 2001.

[17] P. Paillier. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *EUROCRYPT'99*, volume 1592 of Lecture Notes in Computer Science, pages 223-238. Springer-Verlag, 1999.

[18] J. Camenisch, I. Damgård. Verifiable Encryption and Applications to Group Signatures and Signature Sharing. Available at `http://philby.ucsd.edu/cryptolib/1999/99-08.html`, march 1999.

[19] Michael O. Rabin. How to exchange secrets with oblivious transfer. Cryptology ePrint Archive, Report 2005/187, 1981.

[20] I. Damgård, M. Geisler and J. B. Nielsen, From Passive to Covert Security at Low Cost, *TCC*, pages 128-145, 2010.

[21] D. Malkhi, N. Nisan, B. Pinkas and Y. Sella, Fairplay - Secure Two-Party Computation System, *USENIX Security Symposium*, pages 287-302, 2004.

[22] D. Malkhi, N. Nisan, B. Pinkas and Y. Sella, The Fairplay project. http://www.cs.huji.ac.il/labs/danss/FairPlay

[23] T. Staake, F. Thiesse, and E. Fleisch, "Extending the EPC network: the potential of RFID in anti-counterfeiting," in *Proc. ACM SAC*, 2005.

[24] M. Lehtonen, F. Michahelles, and E. Fleisch, "Probabilistic approach for location-based authentication," in *Proc. IWSSI*, 2007.

[25] The Wireshark Project, A powerful packet analysis utility, http://www.wireshark.org