CHALMERS

Distance-Bounding Grouping-Proof Protocols

Master's Thesis in Computer Systems and Networks

CHRISTOFFER KARLSSON

Department of Computer Science and Engineering
Chalmers University of Technology
Gothenburg, Sweden 2014
Master's Thesis 2014:1

Distance-Bounding Grouping-Proof Protocols

CHRISTOFFER S. R. KARLSSON

Examiner: KATERINA MITROKOTSA

Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden July 2014

Abstract

Radio-frequency identification (RFID) is being increasingly used for different types of applications. These applications need protocols that can provide the service and security needed. Currently, there exist distance-bounding protocols that can be used to set an upper limit on how far away a prover can be of a verifier, and grouping-proof protocols that can create a proof of two or more provers being present at the same time. But there exists no protocol that provides both of these security parameters.

The purpose of this thesis is to investigate existing protocols and use the information found to create a new type of protocol, a distance-bounding grouping-proof protocol, that provides the security of both the distance-bounding and grouping-proof protocols. This was done by doing a research in the area of both types of protocols, and use the research results to design a stronger grouping-proof protocol, and three different distance-bounding grouping-proof protocols. All protocols were analysed in terms of security and privacy. A simulation framework was developed, in which some of the protocols were implemented. They had their efficiency, false rejection, and attack resistance evaluated under different conditions.

The main conclusion of this thesis is that it was possible to design a protocol that provides the security of both distance-bounding and grouping-proof protocols. The simulation results show that the security can be greatly increased with low impact on the efficiency, and that the current method to tolerate noise is not adequate.

Acknowledgements

I would like to thank my examiner and supervisor Katerina Mitrokotsa for all help and guidance during my thesis. I would also like to thank my supervisor Elena Pagnin for all her help.

Christoffer Karlsson, July 2014

Contents

List of Figures

List of Tables

1

Introduction

1.1 Background

Radio-frequency identification (RFID) is a wireless communication technology that, as the name states, uses radio-frequency to transfer data. The technology is used in many different types of applications, where two of the main purposes are identification and tracking of objects.

In the authentication application, the user (prover) often has a physical key in the form of an RFID tag, used as an authentication token to an RFID reader (verifier). In such setting, the reader will communicate with the tag, using an authentication protocol, to get its identity and decide whether the tag, and thus the user, should be approved or not. When it comes to tracking of objects, an RFID tag is attached to the object being tracked, and RFID readers are then used to scan the tag and can log that it has been seen. Tracking objects is often done using bar codes and bar code readers (e.g. U.S Postal Service for mail tracking [1]), a technique to which RFID is superior. RFID has many advantages over the bar code systems, such as that the tag does not have to be in the line of sight of the reader, and that the tag is also able to store more information than a bar code.

RFID tags are generally computationally weak and limited in what computations they can carry out. The RFID tags considered in this thesis are low-cost RFID tags, which are not capable of performing any standard cryptography primitives, but only XORing, pseudo-random number generator, and simpler hash functions. It has been shown though, that elliptic curve cryptography (ECC) can be realised on RFID tags, using a special-purpose processor [2, 3].

Some common situations in which RFID are currently used or could be used are: wireless access control in automobile systems, or even in the ignition of cars;

building access control; to make sure a patient has got its medicine in a hospital; to prove that a patient had a prescription for the medicine sold; in meetings to have a proof of who was there; toll collection to register cars passing by; and many more. To handle these different situations properly and securely, different protocols have been designed. This thesis covers two of these protocols, which are known as distance-bounding and grouping-proof, and will now be briefly described.

Distance-Bounding Protocols A distance-bounding protocol is used to make sure that a prover is in the proximity of a verifier. It allows for an upper limit to be set on the physical distance between a prover and a verifier, and tries to make sure that this cannot be faked.

The concept of distance-bounding protocols was first introduced by Brands and Chaum in 1994 [4]. Their protocol was built by using first a lazy phase, in which the prover and verifier exchange some session data. This is followed by a distance-bounding phase - a rapid-bit exchange - in where the verifier determines the distance between itself and the prover. In the distance-bounding phase, short (one bit) challenge messages are sent to the prover, who then replies using the session data previously exchanged. The prover measures the time it takes between sending a challenge and receiving a response, and uses this to approximate the distance between them, considering that a message cannot travel faster than the speed of light. This is repeated for a number of rounds, in which the verifier also checks that the response is correct, and finally it either approves or rejects the prover.

Grouping-Proof protocols A grouping-proof protocol is used to make sure that two or more provers are scanned simultaneously and thus being present at the same time. It is designed so that the verifier will create a proof of the two provers being present at the same time. A proof that then can be verified at some later time. It shall also make sure that it is not possible to fake the presence of a prover.

In 2004, Juels introduced the “yoking-proof” protocol in [5]. The protocol was designed to create a proof of the fact that exactly two provers were present at the same time. This laid the foundation for the simultaneous scanning protocols. A year later, in 2005, Saito and Sakurai further developed the protocol by Juels and presented a “grouping-proof” protocol [6]. This grouping-proof protocol, unlike the yoking-proof protocol, creates a proof of *two or more* provers being scanned simultaneously at the same time. The main idea of this type of protocols is that the verifier starts by generating some random data that is then sent to one of the provers, the prover then uses the received data and computes his response and replies; this reply is then known as that prover’s partial proof. This is then repeated for all the provers within some pre-defined time delta, and finally the verifier creates the proof by storing all these provers partial proofs.

1.2 Problem Description

Many of the protocols that have been proposed as solutions for the distance-bounding are vulnerable to many of the known attacks, some with higher probability of being successful than others. The grouping-proof protocols have also been shown vulnerable to a set of attacks, and new attacks seem to keep coming. The research on protocols that uses ECC is also fairly new and unexplored.

There is no protocol that combines the distance-bounding and grouping-proof protocols, into one protocol that gives the security of both these types of protocols. This type of protocol could be used when even higher security is needed; when insurance of having all provers present at the same time *and* being in the close proximity of a verifier is necessary. One such case could be safety deposit boxes, where the bank official's key and the customer's key needs to be present at the same time and within the proximity of the reader, to open the safety deposit box.

1.3 Purpose and Limitations

The purpose of this study is to analyse existing distance-bounding and grouping-proof protocols, and then use the information to create distance-bounding grouping-proof protocols. The protocols that are created will be of two types:

- A basic type that is limited to work with the computational restrictions of a regular RFID tag, which only allows it to use computations such as XOR, pseudo-random number generators, and simple hash functions.
- A more advanced type that has the same restrictions as the basic type, but is allowed to use ECC, which has been shown possible to use on RFID tags using a special-purpose processor [2, 3].

The created protocols should be secure against the known attacks. For attacks where it is possible, any vulnerability that can be exploited to perform an attack should be disabled. Where it is not possible to entirely disable an attack, the probability to succeed should be as low as possible.

The protocol should also aim to preserve the privacy of the provers as far as possible.

1.4 Contribution

This thesis provides a new type of protocol: the distance-bounding grouping-proof (DBGP) protocol. The DBGP protocol is a combination of the distance-bounding and grouping-proof protocols and aims to achieve the security of both.

Three DBGP protocols have been designed. Two basic types that can be implemented on regular RFID tags, and a more advanced and secure type that makes use of the ECC, which needs a special-purpose processor to be implemented on an RFID tag.

This thesis also proposes a new grouping-proof protocol that is built using the ECC. This protocol is shown to be more secure than other ECC protocols that have been proposed in the literature.

The thesis also provides a simulation framework that can be used to implement and simulate protocols. The framework simulates an RFID reader and RFID tags, and allows to test and evaluate protocols under different conditions, e.g., noise in the communication channel, attacks against the protocol.

1.5 Method

The work of this thesis has been worked out as follows:

1. An initial research was done to find already existing distance-bounding and grouping-proof protocols, as well as to investigate known vulnerabilities and attacks against such protocols. The found protocols were then analysed with respect to functionality, security, and privacy.

2. An ECC based grouping-proof protocol was created, to be used when designing the advanced distance-bounding grouping-proof protocol.

3. Two basic and one advanced distance-bounding grouping-proof protocols were designed. Their designed was based on the analysed, existing protocols found in the initial research, and combined the useful parts of them, and also by using the designed ECC based grouping-proof protocol.

4. The designed protocols were then analysed in terms of security and privacy, to see how they resist possible attacks against them.

5. Finally, the protocols were tested and evaluated in a simulated environment. They had their efficiency, false rejection, and attack resistance tested under different conditions.

1.6 Outline

In Chapter 2, all the technical details that are needed to be known for the reader to understand the thesis are described. The chapter starts by listing all notations that are used throughout the report. Following that, a deeper introduction to the distance-bounding and the grouping-proof protocols is given, where the basics of the protocols, attacks, etc., is described. The choices made on which protocols that will be used in this thesis are then justified, and the protocols themselves are then described in detail.

Chapter 3 starts by describing the methods that have been used when designing the protocols. Following that are all the, in this thesis, proposed protocols and their respective security and privacy analysis presented.

In Chapter 4, the simulation framework that has been built is presented. Following that is a description of the test system and the protocols that will be simulated. Finally, the results from evaluating the protocols in the simulation experiments are presented.

Chapter 5 presents a discussion, on both the used method and the results, as well as on possible future work. The thesis then ends with some conclusions, presented in Chapter 6.

2

Technical Background

This chapter provides the technical background on which this thesis is based on. The chapter begins by presenting the basics of the distance-bounding and grouping-proof protocols, as well as attacks and other considerations that affects them. The chapter ends with a discussion about the protocols that have been chosen for this thesis, and a detailed description of these protocols.

2.1 Notations

This section presents the common notations that will be used when describing protocols in the rest of the thesis. Some notations might be slightly different than how they are presented in the original papers. This is to keep a consistent use of the notations in this thesis.

Common Notations

k	A shared key between a verifier and a prover.
r	Randomly generated value.
c	A counter value.
$\oplus$	Bit-wise XOR function.
$h[M]$	A pseudo-random function, outputs a hash of M.
$MAC[M]$	MAC function, outputs MAC of M.
$PRNG[M]$	Pseudo-random number generator.
$F_k[M]$	A function F, keyed using key k.
ID	Identifier of some prover.

Distance-Bounding Specific Notations

n	Number of rapid-bit exchange rounds.
i	Denotes the round i of the distance-bounding phase.
c_i,r_i	Challenge and response for round i, respectively.
Out_v	Result of the protocol.
B	Probability calculation considering noise.

Grouping-Proof Specific Notations

m	Number of provers in proof.
j	Denotes the j-th prover.
Nun	Lightweight hash function, used to create pseudonyms [7].
T_j	Prover j.
$P_{1,..,m}$	A proof generated for provers 1,...,m.
b_j	Partial proof for prover j.
$[k\|c]_{j\_v}$	Key or counter of prover j, stored in verifier.
$[k\|c]_{j\_ori}$	First initiated value (original) of key or counter of prover j.

ECC Notations

P	Base point (generator).
aP	Scalar multiplication of a and point P.
x_j,$X_j = x_jP$	Private and public key for prover j.
y,$Y = yP$	Private and public key for verifier.
$r_{j,x}$	Random number x, for prover j.

2.2 Distance-Bounding Protocols

This section will give a brief introduction to the basic techniques that are used to build a distance-bounding protocol. Common attacks and how a noisy channel affects the security of a protocol will also be presented.

2.2.1 Protocol Basics

A popular protocol that shows the general concept of how a distance-bounding protocol works, was presented by Hancke and Kuhn in [8]. The HK-protocol was proposed in 2005 and has been shown vulnerable to attacks [9]. Later on, more secure protocols have been proposed [10–12]. However, the HK-protocol is a good example to present the general idea of distance-bounding protocols. The protocol is presented in Figure 2.1, and a detailed description of the protocol is presented below.

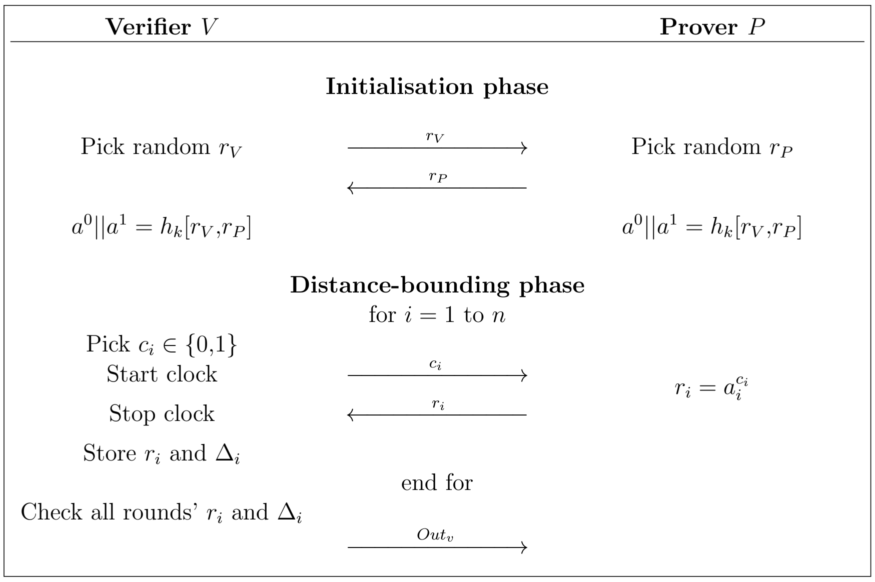


Figure 2.1: The distance-bounding protocol proposed by Hancke and Kuhn.

Description The protocol begins with the verifier generating a random nonce r_V and sends to the prover, which in turn generates its own random nonce r_P and sends it back to the verifier. Then, both the verifier and the prover generates the response vector a using the pseudo-random function, keyed with their shared secret k, on the nonces: $a = h_k[r_V,r_P]$. The response vector is split into two response vectors a^0 and a^1, each of equal length: $|a^0| = |a^1| = n$.

The distance-bounding phase runs for a pre-defined, n, number of times. In each of these rounds, the verifier starts by picking a random challenge value $c_i \in \{0,1\}$, starting its clock, and sends the challenge c_i to the prover. When the prover receives the challenge it responds with $r_i = a_i^{c_i}$, i.e. the i-th bit of the response vector that matches c_i. Then, when the verifier receives the response, it stops the clock and calculates the time delta Δ_i. It then stores the received value r_i and the Δ_i. When all rounds are done, the verifier checks if r_i, for each round i, has the expected value and that each round's Δ_i is less than or equal to a pre-defined maximum value Δ_{max}. Finally, based on the verification step, the verifier will output 1 if the protocol succeeded, else 0.

2.2.2 Attacks on Distance-Bounding Protocols

The most common known attacks against distance-bounding protocols are:

Distance Fraud A malicious prover tries to convince the reader that it is closer than it actually, to pass the distance-bounding protocol without being close enough.

Mafia Fraud An adversary relays messages between a far away honest prover and a verifier, trying to convince the verifier that the prover is closer than it actually is [13].

Distance Hijacking An adversary makes use of honest provers to make the verifier accept a dishonest prover [14].

Terrorist Fraud An adversary collaborates with a dishonest prover to some extent, in order for the adversary to pass the protocol without revealing the secret key [13].

All these attacks aims to pass the distance-bounding phase of the protocol, and since an adversary can try to guess the responses, protocol designers try to lower the probability for any of the attacks to succeed.

2.2.3 Calculating Attack Probabilities

The probability of an attack to succeed against a distance-bounding protocol is what defines how secure or insecure the protocol is against the attack. Therefore, to analyse the protocols, some methods to calculate the probability must be used. The methods to calculate the probability for distance fraud, mafia fraud, and terrorist fraud are therefore presented here. The methods presented here calculate the probability to pass the distance-bounding in one round. This value can then be used to calculate the overall probability of passing all rounds in the attack, by simply raising it to the power of n, or, when considering noisy channels, follow the instructions in Section 2.2.4.

Distance fraud

For a malicious prover P^* to be able to pass the distance-bounding part of the protocol and have the verifier believe it is closer than it is, it must send responses in advance to the verifier. This is done by having P^* guessing the challenge that will be received, calculating the response, and then sending the response to the verifier in advance, i.e. before P^* actually has received the real challenge.

To calculate the probability for a distance fraud to succeed, the probabilities to successfully pass a distance-bounding round when the responses takes certain values are calculated, and then the overall probability is calculated using these separate case probabilities.

Mafia fraud

In a mafia fraud, an adversary has two ways to get the correct response r_i for some round i, which are:

1. Guess the challenge c_i and send to prover P, who then will answer with the response r_i for the sent c_i, which is then replayed to the verifier V.
2. Guessing the round's response r_i and send to V.

An adversary may use both these cases by first guessing the challenge c_i, send to P, get the response r_i and then replay it to V, if c_i was correctly guessed. If the guessed c_i was wrong, the received r_i is discarded and the adversary must instead guess r_i and send it to V.

To calculate the total probability, the two cases are added: $Pr[\text{MF}]_i = Pr[\text{case 1}] + Pr[\text{case 2}]$, which gives the probability to succeed in one round. The total probability to succeed in a mafia fraud attack over all rounds is then $(Pr[MF]_i)^n$.

Terrorist fraud

To analyse the protocols with respect to the terrorist fraud, it is first determined on the design whether they allow for a terrorist fraud to be mounted. If an illegitimate prover P^* is able to give the entire response vector to an adversary A, without A being able to recover P^*'s secret key, the probability to succeed in a terrorist fraud is per definition 100 %.

2.2.4 Noise

In order to guarantee accurate distance-bounding, a communication channel with minimal latency and no error correction codes is used. With no error correction codes, the protocol must, in a practical use, allow for some rounds to be incorrect due to the noise in the channel, i.e. bit errors [8].

Allowing for some rounds to be incorrect will imply that an adversary would only have to guess those τ out of n rounds that must be correct, instead of all n rounds. Therefore, the probability calculations presented in Section 2.2.3 need to be slightly modified when calculating the probability to pass all distance-bounding rounds. When allowing for errors, the binomial distribution $B(n,\tau,q)$ in Equation 2.1 is used, which gives the probability of successfully guessing τ of n rounds, given that one round is guessed correctly with probability q.

$$B(n,\tau,q) = \sum_{i=\tau}^{n} \binom{n}{i} q^i (1-q)^{n-i} \tag{2.1}$$

It has also been shown by Hancke in [15], that allowing for errors in the distance-bounding phase opens up for terrorist frauds against some protocols. It affects the protocols that masks their response vector with their secret key, as a malicious prover simply could give τ bits of the response vector. That would not reveal the secret key, but still allow an adversary to pass.

2.3 Grouping-Proof Protocols

In this section, the basic methods that are mainly used when designing grouping-proof protocols will be presented. Following this, the known attacks against grouping-proof protocols, as well as guidelines that have been developed to avoid or resist these attacks, are presented.

2.3.1 Protocol Basics

To give a simple overview of how a grouping-proof protocol is designed, the protocol proposed by Bolotnyy and Robins in [16], seen in Figure 2.2, will be described. Although the protocol has been shown to have vulnerabilities [17], it still gives an easy understanding of the grouping-proof protocol basics. Following the figure is a detailed description of how the protocol is carried out.

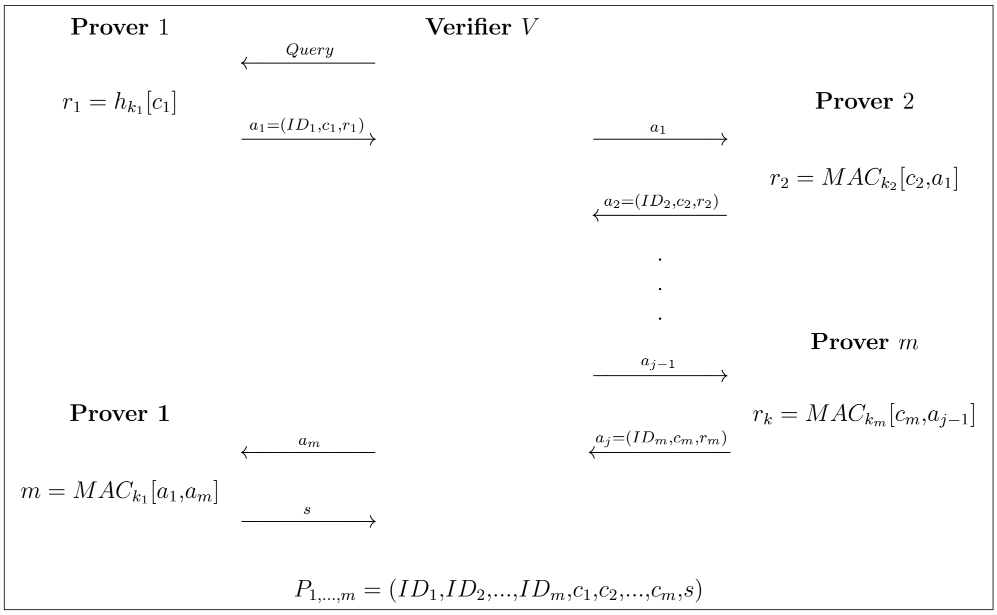


Figure 2.2: The grouping-proof protocol proposed by Bolotnyy and Robins.

Description The protocol begins with the verifier querying some prover, which will be denoted as the first prover (this does not need to be any special prover). This first prover computes $r_1 = h_{k_1}[c_1]$ and sends $a_1 = (ID_1,c_1,r_1)$ back to the verifier.

The verifier then sends a_1 to the next prover, which then computes $r_2 = MAC_{k_2}[c_2,a_1]$ and responds with $a_2 = (ID_2,c_2,r_2)$ back to the verifier. This step is repeated for all m provers that are present in the proof, such that the verifier sends a_{j-1} to prover j, which then computes $r_j = MAC_{k_j}[c_j,a_{j-1}]$ and responds with $a_j = (ID_j,c_j,r_j)$.

When the last prover, ID_m, has been scanned, the verifier sends a_m to the first prover. The first prover then computes $m = MAC_{k_1}[a_1,a_m]$ and sends to the verifier. The verifier now generates the proof $P_{1,...,m} = (ID_1,ID_2,...,ID_m,c_1,c_2, ...,c_m,s)$ of all provers identities, counters and the last generated s. The proof can then be verified by some verifier that has access to all the provers secret keys, by simply doing the same calculations in the same order of the provers.

2.3.2 Attacks on Grouping-Proof Protocols

In recent years, many grouping-proof protocols have been proposed [5–7, 16–24]. The articles proposing new protocols have often found vulnerabilities, privacy issues or new attacks against previously proposed protocols. The most important attacks that have emerged from this are (where anonymity, traceability, and forward security attacks concern privacy of the protocols):

Replay Attack Valid data that has been used in previous sessions is reused maliciously by repeating it, acting as the other part when creating a proof [6, 18].

Subset Replay Attack Maliciously generating a fake proof by using a portion of a legitimate proof [25].

Multiple Impersonation Attack To impersonate a prover, an adversary can eavesdrop a legal run of a protocol and save the created proof. This proof can then be used to create a new, malicious proof between one of the provers in the legal proof and any other prover that was not in the original proof [7, 25].

Anonymity Attack If provers send their identifiers in public, they violate the privacy and an adversary can simply see the provers' identities.

Traceability Attack For provers that send their identifiers in public, and also provers that use some pseudonym that does not change instead of their identifier, an adversary is able to associate a physical prover to some identifier or pseudonym. This is also a violation of a prover's privacy [7, 25].

Forged Proof This attack is performed by eavesdropping the communication of a prover in a proof session. That information can then be used to impersonate the prover in some other session, i.e. it leaks sensitive information that can be re-used [25].

Forward Security If some prover is physically compromised, i.e. an adversary gets hold of its private key, a breach on forward security then allows an adversary to identify and track the prover in historical proof sessions [17].

Race Conditions A prover may store temporary data in its memory during a proof session while waiting for new instructions from the verifier, information that is to be used again in the proof. If a prover then interacts with two or more verifiers, a race condition can occur, as the prover does not know to which of the verifiers the temporary data belongs to [26].

Denial of Proof (m-DoP) By putting some malicious provers among the legitimate ones, an adversary can have the grouping-proof invalid. m-DoP stands for a denial of proof attack made by m malicious provers [17].

Denial of Service (DoS) Trying to make a prover useless. This attack can be launched against provers that have to be synchronised in some way with the verifier. For instance, this is the case when keys are updated in each round.

2.3.3 Guidelines to Resist Attacks

Along with the attacks being presented, many papers have also presented guidelines that should be followed to protect the protocol against being vulnerable to certain attacks. The guidelines that have been presented for the attacks are:

Replay Attacks To protect against replay attacks, each session must be unique. This can be achieved by letting the verifier generate some nonce on which the proof is based on. To protect the nonce from being predicted in any way, it is suggested that the nonce is generated by letting the verifier encrypt a timestamp [25].

Subset Replay Attacks and Multiple Impersonation Attacks To avoid an adversary to mount a subset replay attack, there must be a dependency between the provers in the proof, that is also dependent on the current proof session. This is done by making a prover's partial proof depend on some value from another prover. This value should be unique for that session, for instance it can depend on the verifier's first nonce [25].

Anonymity Attacks To have a prover anonymous, its ID can not be sent in clear to the verifier. Instead, a pseudonym that can identify the prover should be used, which may encrypt or hide the ID in some way. Using a static pseudonym, however, may still allow traceability attacks [25].

Traceability Attacks To protect against an adversary being able to track a prover, the prover needs to use some dynamic pseudonym for identification to the verifier instead of some static identifier [7].

Forged Proofs To disallow an adversary from forging proofs, the protocol must be carefully designed. For instance, running the protocol must not leak any information that an adversary may use to create an own, modified proof in any way [25].

Forward Security A protection against an attack on forward security can be achieved by updating the key that is shared between a prover and a verifier in each iteration of the protocol. This disallows an adversary from regenerating old keys [17].

Race Conditions A race condition can be avoided by simply not having the prover store any data during a proof session, and instead just do static calculations based on the incoming data [17].

Denial of Proof (m-DoP) When using an online (which is assumed in this thesis), trusted verifier, the verifier can authenticate each prover during the proof session. The verifier will then be able to discover a fake prover immediately and thus disallow it from breaking the proof [17].

Denial of Service (DoS) The easiest way to protect against DoS attacks is by not having any need of synchronisation between the verifier and the provers. However, when they need to be synchronised in some way, some data can be sent along the messages that allows them to synchronise, such as a counter.

2.3.4 Current State of Grouping-Proof Protocols

Table 2.1 presents the current state of the grouping-proof protocols' resistance to the known attacks against them. "Y" denotes that the protocol is secure against the attack, "N" denotes that the protocol is not secure against the attack, and "-" denotes that the state is unknown.

	Replay	Subset	Multiple Impersonation	Anonymity	Traceability	Forged Proof	Forward Security	Race Conditions	m-DoP	DoS
Juels [5]	N [25]	Y [25]	N [25]	N [25]	N [25]	Y [25]	N [17]	N [17]	N [17]	N [25]
Saito and Sakurai [6]	N [25]	N [25]	N [25]	Y [25]	Y [25]	Y [25]	N [17]	N [17]	N [17]	N [25]
Bolotnyy and Robins [16]	Y [25]	N [25]	Y [25]	Y [25]	Y [25]	Y [25]	N [17]	Y [17]	Y[1] [17]	N [25]
Piramuthu [18]	Y [25]	Y [25]	Y [25]	N [25]	N [25]	Y [25]	N [17]	N [17]	N [17]	N [25]
Peris-Lopez et al. [7]	Y [25]	Y [25]	Y [25]	Y [25]	Y [25]	Y [25]	N [17]	N [17]	N [17]	N [25]
Cho et al. [19]	Y [25]	Y [25]	Y [25]	N [25]	N [25]	Y [25]	-	-	-	N [25]
Lien et al. [20]	Y [25]	Y [25]	Y [25]	N [25]	N [25]	Y [25]	N [17]	Y [17]	N [17]	N [25]
Burmester et al. [21]	Y [25]	Y [25]	N [25]	Y [25]	Y [25]	Y [25]	-	-	-	Y [25]
Chien and Liu [22]	Y [25]	Y [25]	Y [25]	Y [25]	N [25]	Y [25]	-	-	-	Y [25]
Huang and Ku [23]	Y [25]	Y [25]	Y [25]	N [25]	N [25]	N [25]	-	-	-	N [25]
Chien et al. [24]	Y [25]	Y [25]	Y [25]	N [25]	N [25]	N [25]	-	-	-	N [25]
Lo and Yeh [17]	Y [17]	Y [17]	Y [17]	Y [17]	Y [17]	Y [17]	Y [17]	Y [17]	Y [17]	Y [17]
ECC-GP[2]	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y

[1] Can resist when $m = 1$, i.e. one malicious tag.

[2] Proposed in this thesis, in Section 3.4.

Table 2.1: Current state of grouping-proof protocols attacks resistance.

2.4 Justifying Choices

This section will describe which protocols that have been identified as more suitable for the design of a distance-bounding grouping-proof protocol. A brief description justifying each chosen protocol will also be given.

Distance-Bounding Protocols

The Modified Swiss-Knife protocol, presented by Fischlin and Onete in [11], and the SKI protocols, presented by Boureanu et al. in [12], are considered to be among the latest and most provable secure protocols in distance-bounding [27]. They are both light-weight and can be implemented on RFID tags. They are built slightly different and provide different security against the known distance-bounding attacks.

The most recent distance-bounding protocol that builds on ECC is presented by Hermans et al. in [28]. Besides that it is built around the ECC technology, the main difference in this protocol compared to the previously mentioned protocols, is that it provides strong privacy. The provers are both anonymous and untraceable for any adversary.

Grouping-Proof Protocols

Among all the presented grouping-proofs that have been presented, the protocol presented by Lo and Yeh in [17] seems to be one of the most secure. It follows the guidelines that have been presented and claims to be secure against all the known attacks against grouping-proof protocols. The protocol is also fairly easy to follow, as it follows the general grouping-proof protocol design.

There are also grouping-proof protocols presented that are built using ECC [29, 30], in which the latest and most secure seems to be the protocol presented by Hermans and Peeters in [30]. However, I have chosen not to use this protocol. The reason for this is that it barely follows the guidelines that have been created for grouping-proof protocols (presented in Section 2.3.3). The most critical about this, in my opinion, is that it does not chain the provers' partial proofs. This makes all provers' partial proofs independent, as they are generated from random numbers and a value from the verifier, and has nothing that depends on the other provers. For a protocol to be secure, it needs to chain the partial proofs [25].

Instead, I will propose a new grouping-proof protocol that makes use of the ECC, which will be used to create a more secure distance-bounding grouping-proof protocol. The protocol will be built based on the ECC methods seen in the distance-bounding protocols, and it should follow the guidelines for grouping-proof protocols.

2.5 Chosen Protocols

This section presents in detail the protocols that have been chosen as the base when designing the distance-bounding grouping-proof protocols.

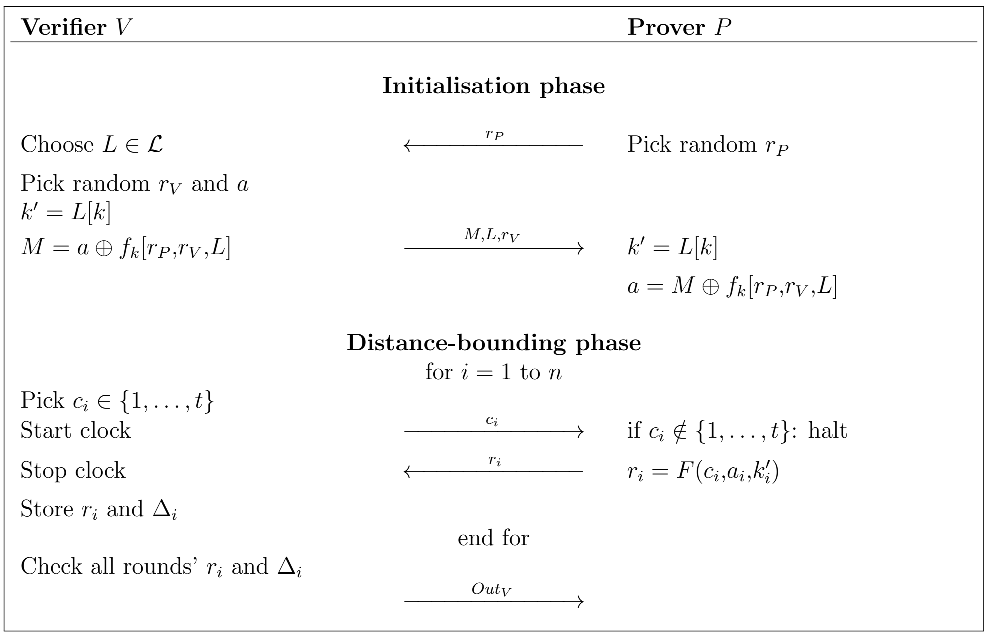
2.5.1 The SKI Protocols

The SKI protocols are different lightweight protocols that all use the same protocol structure, but with different settings to achieve different levels of security. First, the structure of the protocol will be described, then the setting of the protocol that will be used.

Protocol Structure

The general protocol structure of the SKI protocols is presented in Figure 2.3, and a more detailed description is presented after the figure. It is slightly more advanced than the basic protocol that was presented in Section 2.2.1 and does thus have some more, special notations that varies between different settings, it makes use of:

L,$\mathcal{L}$	L is a chosen linear transformation from a set $\mathcal{L}$.
t	The number of choices of challenges.
F	Response function, used to calculate the response.



Verifier V		**Prover** P
	Initialisation phase	
Choose $L \in \mathcal{L}$	$\xleftarrow{r_P}$	Pick random r_P
Pick random r_V and a		
$k' = L[k]$		
$M = a \oplus f_k[r_P,r_V,L]$	$\xrightarrow{M,L,r_V}$	$k' = L[k]$
		$a = M \oplus f_k[r_P,r_V,L]$
	Distance-bounding phase	
	for $i = 1$ to n	
Pick $c_i \in \{1,\ldots,t\}$		
Start clock	$\xrightarrow{c_i}$	if $c_i \notin \{1,\ldots,t\}$: halt
Stop clock	$\xleftarrow{r_i}$	$r_i = F(c_i,a_i,k'_i)$
Store r_i and Δ_i		
	end for	
Check all rounds' r_i and Δ_i		
	$\xrightarrow{Out_V}$	

Figure 2.3: The SKI distance-bounding protocols' general structure.

Initialisation phase Before the steps that are seen in Figure 2.3 are carried out, it is assumed that the verifier gets the identity of the prover, in order to be able to use the correct shared key k.

The protocol begins with the prover generating a random nonce r_P and sends it to the verifier. The verifier then chooses a linear transformation from a predefined set of transformations, generates a nonce r_V and the response vector a. A second key k' is computed by applying the chosen transformation to the shared key k. The verifier then prepares a message M, containing the response vector a encrypted by XORing the output of the keyed one-way function f_k with a. Finally, the encrypted message M, the chosen transformation L, and the nonce r_V is sent to the prover. The prover uses the received transformation to also compute k', and decrypts the message M to obtain the response vector a.

Distance-bounding phase The verifier starts its clock and chooses a challenge c_i from the set of possible challenges $\{1,...,t\}$ and sends this to the prover. The prover will then first check that the challenge is an allowed challenge. If the challenge is invalid, it will abort. Else, the prover computes the response r_i using the response function $F(c_i,a_i,k'_i)$ and sends the result back to the verifier. When the response is received, the verifier stops its clock and stores the time delta Δ_i and response r_i for the round. The described steps here are repeated for n times.

Verification When the distance-bounding phase is over, the verifier checks that at least τ of the responses were correct and that none of them took longer time than Δ_{max}.

Settings

The two main instances of the SKI protocols presented in [12] are SKI_{lite} and SKI_{pro}. Both are considered lightweight, but they differ in execution and on the security level they provide. Since the RFID devices considered in this thesis have very limited computation, the *lite* setting of the protocol, which is more lightweight than the *pro* setting, is more interesting and will only be used. Readers interested in the other setting are referred to [12].

The SKI_{lite} does however not offer any protection against terrorist fraud. Because of that, neither of the transformed key k' nor the transformation L (i.e. $L = \emptyset$) are used. The other settings in SKI_{lite} are:

$q = 2$	Only using bits.
$t = t' = 2$	The challenge is considered being a bit as well.
$F = F_{\text{XOR}}$	With $t' = 2$ and not using k', the response function will work as $F(c_i,a_i) = a_i^{c_i}$.

2.5.2 The Modified Swiss-Knife Protocol

The Modified Swiss-Knife Protocol presented in [11] by Fischlin and Onete, is a protocol that basically extends the original Swiss-Knife protocol [10], by using a second key k', which makes it, as they present it, GameTF-secure.

GameTF-security can be simplified as: *An adversary A that runs a terrorist fraud with a verifier V and a prover P^* will have an end view V of the run, containing all known data from the run. For a protocol to be GameTF-secure, this end view V should not be of any help to another adversary A' that runs a mafia fraud attack on V and P.*

The protocol is presented in Figure 2.4 and is followed by a more detailed description.

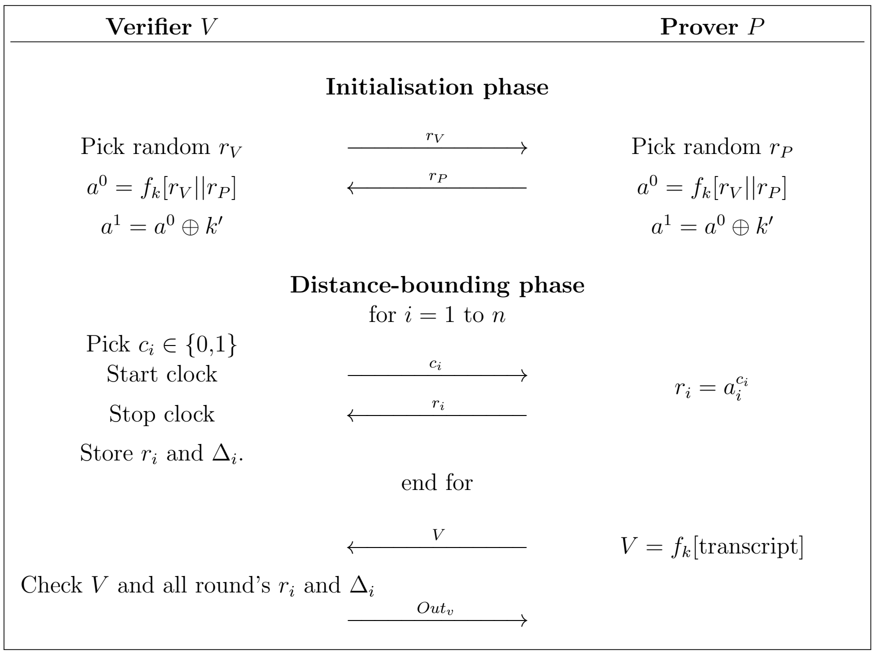


Figure 2.4: The Modified Swiss-Knife (MSK) distance-bounding protocol.

Initialisation phase As with the SKI protocols, it is also assumed here that, before the protocol runs, the verifier gets the identity of the prover in order to be able to use the correct shared keys k and k'.

The protocol begins with the verifier generating a random number r_V, which is then sent to the prover. Upon reception of r_V, the prover generates a random number r_P and sends back to the verifier. Following this, they both compute the response vectors $a^0 = f_k[r_V||r_P]$ and $a^1 = a^0 \oplus k'$.

Distance-bounding phase For n number of rounds, the verifier picks a challenge $c_i \in \{0,1\}$, starts a clock and sends c_i to the prover. The prover responds to the challenge immediately by taking the i-th bit of the response vector matching the challenge: $r_i = a_i^{c_i}$. When the verifier receives the response, it stops the clock and stores the response r_i and time delta Δ_i.

Verification When the distance-bounding phase is over, the prover will send a hash of the transcript (all messages carried out through the protocol session) $V = f_k[\text{transcript}]$ to the verifier. The verifier will then check the validity of the received transcript and also check so that, for each round, the response is correct and that it took at most Δ_{max}. Following this, it will output whether the prover is accepted or not.

2.5.3 The Efficient Online Verifier Based Protocol

The Efficient Online Verifier Based Protocol (EOBVP) was proposed by Lo and Yeh in [17]. The protocol runs different procedures for the first tag (which can be any tag) and all the other tags, and is therefore separated and presented in different figures. Figures 2.5 shows how it is run for the first prover and Figure 2.6 shows how it is run for the rest of the provers. Following the figures is a more detailed description of the protocol. The description here is a bit different from how it is presented in the original article, since it in this case is assumed that the prover and verifier are the same entity.

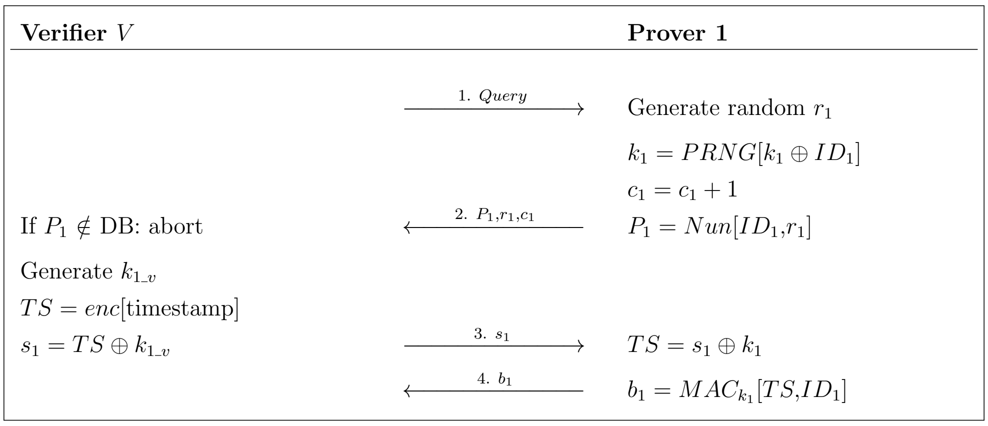


Figure 2.5: The Efficient Online Verifier Based (EOVBP) grouping-proof protocol for the first prover.

The first prover In Figure 2.5, the proof generation step for the first prover is seen, and it works as follows:

Step 1 The verifier sends a *Query* to one of the provers, which, in this session, will be the first prover.

Step 2 When the prover receives the *Query*, it will generate a random number r_1, update its key using $k_1 = PRNG[k_1 \oplus ID_1]$, update its counter value $c_1 = c_1 + 1$ and calculate its pseudonym $P_1 = Nun[ID_1,r_1]$. The prover then sends a response containing P_1,r_1, and c_1 back to the verifier.

Step 3 *Verification*: The verifier identifies the prover by computing $Nun[ID_j,r_1]$ for all provers that are stored in its database and compares each calculated value with P_1. If the verifier does not find a match, the proof session is terminated. Else, it moves forward.
Get prover's current key: To derive the key, the verifier compares the stored counter value c_{1_v} with the received value c_1. If $c_{1\_v} > c_1$, then the verifier needs to compute them from their initial values, i.e. set $c_{1\_v} = c_{1\_ori}$ and $k_{1\_v} = k_{1\_ori}$. Then, to generate the current key, the verifier performs the key update computation $k_{1\_v} = PRNG[k_{1\_v} \oplus ID_1]$ and $c_{1\_v} = c_{1\_v} + 1$, and does these computations $c_1 - c_{1\_v}$ times, so that $c_{1\_v} = c_1$, which is also when $k_1 = k_{1\_v}$.
Generate response: The verifier reads the current timestamp and encrypts it $TS = enc$[timestamp], and calculates the response value $s_1 = TS \oplus k_{1\_v}$. The verifier then sends the encrypted s_1 to prover 1 and starts a time out mechanism.

Step 4 When the prover receives s_1 it uses its own key to decrypt TS through $TS = s_1 \oplus k_1$. The prover then computes its partial proof $b_1 = MAC_{k_1}[TS, ID_1]$, and finally responds with b_1 to the verifier.

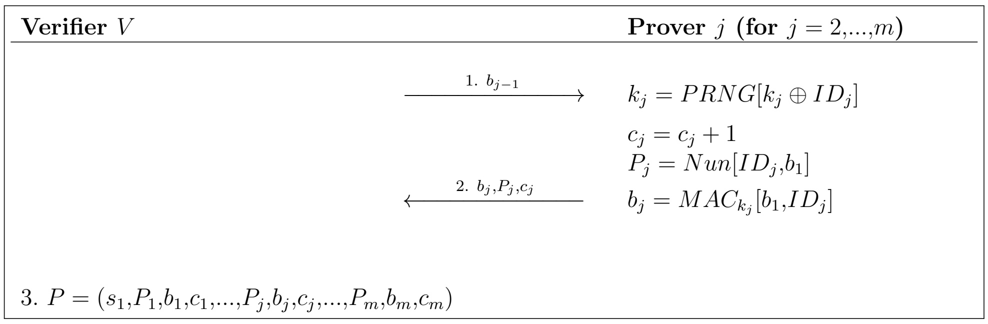

Verifier V
Prover j **(for** $j = 2,...,m$**)**
1. b_{j-1}
$k_j = PRNG[k_j \oplus ID_j]$
$c_j = c_j + 1$
$P_j = Nun[ID_j,b_1]$
2. b_j,P_j,c_j
$b_j = MAC_{k_j}[b_1,ID_j]$
3. $P = (s_1,P_1,b_1,c_1,...,P_j,b_j,c_j,...,P_m,b_m,c_m)$

Figure 2.6: The Efficient Online Verifier Based (EOVBP) grouping-proof protocol for provers 2 to m.

Prover j (for $j = 2,...,m$) In Figure 2.6, the proof generation step for all the other provers 2 to m is seen, and it works as follows:

Step 1 The verifier sends b_1 to prover j.

Step 2 The prover updates its key $k_j = PRNG[k_j \oplus ID_j]$ and its counter value $c_j = c_j + 1$. Then it calculates its pseudonym $P_j = Nun[ID_j,b_1]$ and its partial proof $b_j = MAC_{k_j}[b_1,ID_j]$. Finally, it sends a response to the verifier that contains the data P_j,b_j, and c_j.

Proof verification When the session is finished and the proof $P_{1,...,m}$ is created, the verifier will verify by performing the following steps:

Step 1 Verify that the time for the proof generation is less than some predefined time period Δ_2, i.e. (current timestamp $- TS) \leq \Delta_2$.

Step 2 *Identify provers*: To identify and find all provers that was present in the proof, the verifier calculates $Nun[ID_j,b_1]$ for all provers stored in database and compare against all P_j in the proof $P_{1,...,m}$.
Verification: The verifier derives the key for all provers in the same way that was done in step 3 of the first prover. Once the key is derived, the verifier calculates $MAC_{k_{j\_v}}[b_1,ID_{j\_v}]$ and compares it against b_j to verify each prover.

2.5.4 The Hermans-Peters-Onete Protocol

The protocol presented by Hermans et al. in [28], will in this report be denoted as the Hermans-Peters-Onete Protocol (HPO). The HPO protocol is an ECC based distance-bounding protocol. The protocol is presented in Figure 2.7 and a detailed description follows the figure.

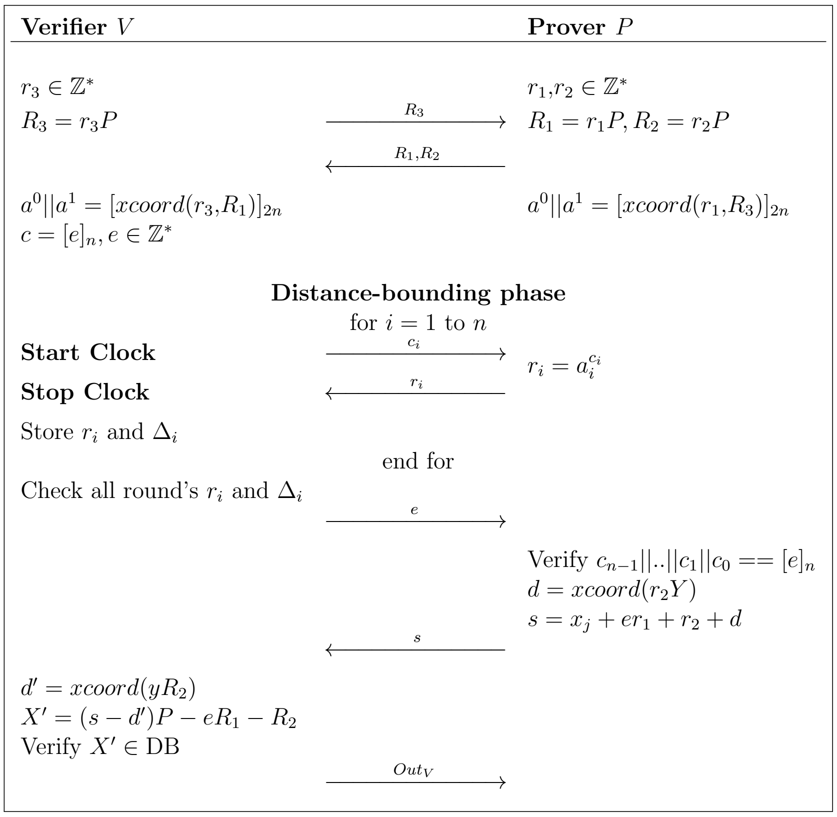


Figure 2.7: The Hermans-Peters-Onete (HPO) distance-bounding protocol.

Initialisation phase The protocol begins with the verifier generating a number $r_3 \in \mathbb{Z}^*$, multiplies the number with the base point P and sends the new point to the prover. The prover then generates two numbers $r_1,r_2 \in \mathbb{Z}^*$, which then are also multiplied with the base point P and are then sent back to the verifier. Following that, both the verifier and the prover takes the x-coordinate out of the point $r_3R_1 = r_1R_3 = r_1r_3P$, using the *xcoord*-function [31]. Then they generate the response vectors by taking the first $2n$ bits of that x-coordinate. The verifier then sets up the challenge vector by taking the first n bits from some randomly selected number $e \in \mathbb{Z}^*$.

Distance-bounding phase For each round i of the distance-bounding phase, the verifier starts a clock and sends the i-th bit of the challenge vector to the prover. The prover responds with the i-th bit of the matching response vector: $r_i = a_i^{c_i}$. When the verifier receives the response, it stops the clock and stores the time delta and response for later verification.

Verification and Authentication After the distance-bounding phase, the verifier checks for each round, that Δ_i is smaller than Δ_{max} and that the received response r_i is correct for the challenge sent.

If the verification was correct, the verifier sends the number e to the prover. Upon reception, the prover verifies that the challenges received matches the n first bits of e. It then takes the x-coordinate of the point $r_2Y = yR_2 = r_2yY$, which is used along with the prover's private key x and previous information, to create a number $s = x + er_1 + r_2 + d$. This s number is then sent to the verifier for authentication.

After receiving s, the verifier computes the same x-coordinate and then applies the same computations done to s, to points, so that it can compute the point $X' = x'P$, i.e. the public key of the prover. The verifier then checks if this point X' is in the database. If it is, then the prover has been verified to be legitimate.

3

Designed Protocols

This chapter will introduce three versions of a new type of protocol: the Distance-Bounding Grouping-Proof (DBGP) protocol. The chapter will begin with a brief description of how the protocols have been designed. Following that is a presentation of the protocols proposed in this thesis, along with security and privacy analyses of them. The presentation of the proposed protocols will start with the two basic DBGP protocols, i.e. the protocols designed to work on normal RFID tags. After that, the ECC based grouping-proof protocol is presented, followed by the ECC based DBGP protocol, designed for more powerful RFID tags.

3.1 Protocol Design Methods

This section will present the general method used when designing the DBGP protocols, security and privacy improvements that have been made to the EOVBP, and a description of how the security and privacy analyses have been carried out.

3.1.1 Combining Protocols

In general, the DBGP protocols have been designed by merging the distance-bounding and the grouping-proof protocols. After the merging of the two different types of protocols, there were things that could be improved to make them more efficient. The efficiency improvements that have been made, have been done on parts of the original protocols that could be merged in the combined DBGP protocol. Things such as sending more data in one message instead of having several messages and re-use data when possible in functions instead of generating new data for each function.

3.1.2 Security and Privacy Improvements

The EOVBP was the only protocol in which the security and privacy could be enhanced, as no issues were found in the other protocols. The security improvement of EOVBP was to fix the lack of prover dependencies, and the privacy improvement was to strengthen the pseudonyms.

Prover Dependencies

In the EOVBP, the partial proof of each prover (for provers $2,...,m$) is computed by $b_j = MAC_{k_j}[b_1,ID_j]$, and the information from this prover that is stored in the proof is P_j, b_j, and c_j. If the proofs are stored for long time and should be available for verification afterwards, the stored proof is vulnerable to modification. An adversary with access to the database could simply delete a prover's entry from the proof (except the first prover), and the modified proof would still be accepted. To prevent this, each prover's partial proof now depends on the previous prover's partial proof: $b_j = MAC_{k_j}[b_{j-1},ID_j]$. This disallow verification if a prover's entry is removed, as verifying prover b_{j+1} is impossible without b_j.

Strengthen the Pseudonyms

In the EOVBP, each pseudonym is computed by $P_j = Nun[ID_j,b_1]$. If there are m provers in a proof, an adversary would only have to do one exhaustive search over all possible values for ID_j, and thus get the ID and pseudonym for all m provers that were in the proof. Therefore, the pseudonym has been changed to be computed by $P_j = Nun[ID_j,b_{j-1}]$, which forces an adversary to do one exhaustive search for each prover it wants to find the ID for, as b_{j-1} will be different for each prover. Thus, the cost of finding the ID for all provers is increased by a factor of the number of provers, m.

3.1.3 Security and Privacy Analyses

All protocols that have been designed have been analysed with respect to the relevant attacks, using the following methods:

Distance-Bounding Attacks To analyse the protocols with respect to the distance-bounding attacks, presented in Section 2.2.2, the methods presented in Section 2.2.3 have been used to calculate the probabilities of the attacks being successful.

Grouping-proof attacks To analyse the protocols with respect to the grouping-proof attacks, presented in Section 2.3.2, the guidelines that were presented in Section 2.3.3 have been used. Each protocol was reviewed for each of the guidelines, to determine to what extent the protocol follows them.

3.2 The Distance-Bounding Grouping-Proof Basic 1 Protocol

In this section, the first Distance-Bounding Grouping-Proof Basic protocol (from now on called DBGP-B1) is presented, as well as a security and privacy analysis of it. The protocol is mainly built on the distance-bounding protocol SKI, in its *lite* setting, presented in Section 2.5.1, and the grouping-proof protocol EOVBP, presented in Section 2.5.3.

3.2.1 Description

The DBGP-B1 protocol, as the grouping-proof protocol it is built on, use slightly different methods for the first prover, compared to the rest of the provers. Therefore, the protocol is described in two figures; Figure 3.1 describes how the protocol is run for the first prover and Figure 3.2 describes how the protocol is run for all other provers that are participating in the proof.

Following the figures is a more detailed description of how the protocol works. Specific details for some of the computations that are the same as in the original protocols have been left out of this description, and the reader is instead referred to Section 2.5.1 and Section 2.5.3 for those details.

Verifier V		**Prover** 1
	Initialisation phase	
	$\xrightarrow{Query}$	Generate random r_1
		$k_1 = PRNG[k_1 \oplus ID_1]$
		$c_1 = c_1 + 1$
If not $P_1 \in$ DB: abort	$\xleftarrow{P_1,r_1,c_1}$	$P_1 = Nun[ID_1,r_1]$
Generate k_{1_v} from c_{1_v}		
Generate $a = a^0\|a^1$		
$TS = enc$[Timestamp]		
$s_1 = TS \oplus k_{1\_v}$		
$M = a \oplus f_{k_1}[r_1,TS]$	$\xrightarrow{s_1,M}$	$TS = s_1 \oplus k_1$
		$a = M \oplus f_{k_1}[r_1,TS]$
	$\xleftarrow{b_1}$	$b_1 = MAC_{k_1}[TS,ID_1]$
	Distance-bounding phase	
	for $i = 1$ to n	
Pick $c_i \in \{0,1\}$		
Start clock	$\xrightarrow{c_i}$	$r_i = a_i^{c_i}$
Stop clock	$\xleftarrow{r_i}$	
Store r_i and Δ_i		
	end for	
Check all rounds' r_i and Δ_t		
	$\xrightarrow{Out_V}$	

Figure 3.1: The Distance-Bounding Grouping-Proof Basic 1 (DBGP-B1) protocol for the first prover.

Verifier V		**Prover** j **(for** $j = 2,...,m$**)**
	Initialisation phase	
	$\xrightarrow{b_{j-1}}$	Generate random r_j
		$k_j = PRNG[k_j \oplus ID_j]$
		$c_j = c_j + 1$
		$P_j = Nun[ID_j, b_{j-1}]$
If not $P_j \in$ DB: abort	$\xleftarrow{P_j, b_j, r_j, c_j}$	$b_j = MAC_{k_j}[b_{j-1}, ID_x]$
Generate k_{j_v} from c_{j_v}		
Generate $a = a^0 \vert\vert a^1$		
$M = a \oplus f_{k_{j\_v}}[r_j, b_{j-1}]$	$\xrightarrow{M}$	$a = M \oplus f_{k_j}[r_j, b_{j-1}]$
	Distance-bounding phase	
	for $i = 1$ to n	
Pick $c_i \in \{0,1\}$		
Start clock	$\xrightarrow{c_i}$	$r_i = a_i^{c_i}$
Stop clock	$\xleftarrow{r_i}$	
Store r_i and Δ_i		
	end for	
Check all rounds' r_i and Δ_t		
	$\xrightarrow{Out_V}$	
When all provers are done:		
$P_{1,...,m} = (s_1, P_1, b_1, c_1, ..., P_j, b_j, c_j, ..., P_m, b_m, c_m)$		

Figure 3.2: The Distance-Bounding Grouping-Proof Basic 1 (DBGP-B1) protocol for provers 2 to m.

Details The verifier initiates a session by sending a *Query* to one of the provers, which will then act as the first prover. The prover generates a random value r_1, updates its key k_1 and its counter value c_1, and generates its pseudonym P_1. The prover then sends P_1, r_1, and c_1 to the verifier, which, upon receipt, verifies that P_1 is in the database. The verifier then computes the shared key k_{1_v}, an encrypted timestamp TS (the "nonce"), and encrypts TS with k_{1_v} into s_1. The response vector a is also calculated, and encrypted with the output of $f_{k_1}[r_1,TS]$ into M. Finally, the verifier sends the encrypted messages s_1 and M to the prover. The prover retrieves TS and a from the encrypted messages, computes its partial proof b_1 and sends it back to the verifier. When the verifier receives b_1, it will begin the distance-bounding phase. For n rounds, the verifier starts a clock and sends a challenge to the prover, which then responds with the response value from the response vector a, matching the challenge, to the verifier. When the verifier receives the response r_i, it stops the clock and stores the time delta Δ_i and the response r_i. After the distance-bounding phase, the verifier will verify that all Δ_i were within the allowed Δ_{max}, and that at least τ responses were correct.

The procedure is almost identical for the rest of the provers. But, instead of receiving a query as the first message, the other provers will get the previous prover's partial proof. A prover then generates all the values as the first prover did, plus its own partial proof, and sends this to the verifier. Now, however, the verifier only generates the challenge vector a, and encrypts with the output of $f_{k_{j\_v}}[r_j,b_{j-1}]$ into M, and sends M to the prover. Since the prover has already sent its partial proof, the protocol continues with the distance-bounding phase in the same way as described for the first prover.

When all provers are finished, the verifier will check that the time taken for the entire proof has note exceeded some pre-defined maximum time. If the time was within the boundaries, the verifier saves all data and partial proofs to its database, from where it later again can be retrieved and verified.

3.2.2 Security and Privacy Analysis

The following section presents the security and privacy analysis of the DBGP-B1 protocol, analysed with respect to the method described in Section 3.1.3.

Distance Fraud

The DBGP-B1 protocol has a challenge $c_i \in \{0,1\}$ and a response $r_i \in \{0,1\}$, where r_i is decided by $a_i^{c_i}$. A malicious prover P^* may calculate the response values for each round in advance, for the two values c_i can take. From this, it follows that there are two cases that can happen in each round:

1. The first case, which also is the best case that can happen for P^*, is when $a_i^0 = a_i^1$, as the response is the same whatever the challenge is. Since r_i can take only two values, it is easy to see that the probability for this to happen is $\frac{1}{2}$, as we get: $Pr[\text{case 1}] = 1 * \frac{1}{2}$.

2. The other case is when $a_i^0 \neq a_i^1$, which also happens with a probability of $\frac{1}{2}$. This time, P^* must guess what the challenge will be. As the challenge is one of two values, the probability that P^* makes a correct guess is $\frac{1}{2}$. The probability for this case to happen and for P^* to make a correct guess is thus: $Pr[\text{case 2}] = \frac{1}{2} * \frac{1}{2} = \frac{1}{4}$.

This gives a total probability to pass one round: $Pr[\text{DF}]_i = Pr[\text{case 1}] + Pr[\text{case 2}] = \frac{1}{2} + \frac{1}{4} = \frac{3}{4}$, and to pass the distance-bounding phase in all rounds: $Pr[\text{DF}] = (\frac{3}{4})^n$.

Mafia Fraud

As described in Section 2.2.3, there are two ways to get the correct reponse in a mafia fraud:

1. The probability for the first case, guessing the challenge correctly, is $Pr[\text{case 1}] = \frac{1}{2}$, as the challenge can be of only two values.

2. The probability of making an incorrect guess about the challenge is $1 - \frac{1}{2}$, and in this case, the adversary needs to guess the response value and send to the verifier. The probability of guessing the correct response is $\frac{1}{2}$, and thus, it follows that the total probability of succeeding in this case is $Pr[\text{case 2}] = \frac{1}{2} * \frac{1}{2} = \frac{1}{4}$.

The total probability for succeeding in one round is then $Pr[\text{MF}]_i = Pr[\text{case 1}] + Pr[\text{case 2}] = \frac{1}{2} + \frac{1}{4} = \frac{3}{4}$, and to pass the distance-bounding phase in all rounds: $Pr[\text{MF}] = (\frac{3}{4})^n$.

Terrorist Fraud

The protocol does not protect against terrorist frauds. A malicious prover P^* can give away the information needed to pass the protocol, the a vector, without revealing its secret key.

Giving away a allows the adversary to retrieve $f_{k_j}[r_j, b_{j-1}]$, but this will not give the adversary any advantage to pass the protocol in a future session, since:

1. $f_{k_j}[M]$ is a one-way function, producing a hash of M, keyed with k_j. This disallows recovering k_j from the hashed output.

2. $f_{k_j}[r_j,b_{j-1}]$ is unique for this prover and this session only. In any future session, the values r_j and b_{j-1} will be different.

3. a is randomly generated for this distance-bounding phase and will be different in any future run of the protocol.

Replay Attacks

The protocol follows the guidelines and makes use of an encrypted timestamp to have each session unique. This prevents an adversary from predicting the nonce or pre-calculating timestamps. The first prover builds its partial proof b_1 on this timestamp, and every prover that follows builds their partial proof from the partial proof of the prover preceding it. This makes each prover's proof dependent on the timestamp, which is unique for this session, and thus disallows for any data to be replayed in another session.

Subset Replay Attacks and Multiple Impersonation Attacks

The description of how the protocol resists replay attacks also shows that it can resist subset replay and multiple impersonation attacks, as it clearly states that the provers have a dependency between them, and to the current session.

Anonymity Attacks

The protocol follows the guidelines on hiding the identifier of a prover. Instead of sending a prover's real ID, a pseudonym, created with the Nun function is being sent. As the Nun function is a one-way hash function, and the input to it is the ID of the prover, which is unknown for an adversary, there is no way for an adversary to calculate the value and compare.

Traceability Attacks

The pseudonym P_j is created by applying the Nun function to the prover's ID and the random number r_j that has been generated by the prover for this session. This makes the pseudonym of a prover unique for each prover and session, and it can thus not be tracked by an adversary.

Forged Proofs

To impersonate a prover and be able to create a partial proof using that prover's data, an adversary A would need to get possession of the prover's key k_j, counter c_j, and identifier ID_j.

The counter value is needed in order to be synchronised with the verifier and it can easily be achieved as it is sent in clear to the verifier.

The identifier must be known in order to create a valid pseudonym P_j and a valid partial proof b_j. The identifier is never sent in clear, and both P_j and b_j are created using one-way functions. The only way for A to get these values is by brute forcing P_j or b_j, which, given that they are of enough length, is infeasible.

The key must be known to create the partial proof b_j and to be able to retrieve the response vector a. Both of these values are the output of one-way functions, in which k_j is used to create keyed output values. This means that they cannot be reversed, and thus, the only way to retrieve the key is by brute force, which, given that proper hash length is used, is infeasible.

The key is sent within a non one-way function for the first prover in the protocol: $s_1 = f_{k_v}[TS] \oplus k_1$. The only way to retrieve k_1 from s_1 would be if an adversary got hold of the encrypted timestamp. This value, however, is never sent in clear and is only known by the verifier and the first prover, which only makes use of it in a one-way function.

Thus, there is no feasible way to retrieve either a prover's identifier nor the private key.

Forward Security

The protocol supports for partial resistance to forward security. If an adversary gets hold of a prover, there is no way to retrieve the old keys that have been used in previous protocols, as they are updated through a non-reversible pseudo-random number generator.

An adversary gaining access to the prover's stored values will be able to track the prover in historical sessions. The values P_j, b_j, and r_j are all sent in clear and stored with the proof, for the verifier to be able to verify and identify the prover. As the pseudonym P_j is calculated by $P_j = Nun[ID_j,r_j]$, it would then be possible for an adversary to calculate P_j when it is in possession of the prover's ID_j, and thus check a proof for the presence of the prover.

Race Conditions

The protocol is partially resistant to race conditions. The provers do not store any information during the lazy phases, i.e. all phases but the distance-bounding phase. Therefore, it would not be possible for the protocol to be desynchronised during this time. A prover does however store the response vector a, to use in the distance-bounding phase. This is in the nature of the distance-bounding protocols and can not be removed.

Denial of Proof (m-DoP)

Any m-DoP attack against the protocol would fail, since the verifier is in online mode and each prover is verified in the beginning of the verifier-prover communication.

Denial of Service (DoS)

An adversary that attacks a prover and has its key desynchronised with the verifier is not a problem. When the verifier wants to communicate with the prover, the counter allows the key to be updated and synchronised.

3.3 The Distance-Bounding Grouping-Proof Basic 2 Protocol

This section will describe the second Distance-Bounding Grouping-Proof Basic protocol (DBGP-B2), as well as a security and privacy analysis of it. The protocol is built on the distance-bounding protocol MSK, presented in Section 2.5.2, and the same grouping-proof protocol as DBGP-B1 - the EOVBP protocol from Section 2.5.3.

3.3.1 Description

The protocols do, just as DBGP-B1, work differently for the first prover and the rest of the provers. Therefore, this protocol is also described in two figures; Figure 3.3 describes how the protocol is run for the first prover and Figure 3.4 describes how the protocol is run for the rest of the provers. Details on computations that have been described in the original protocols are left out of the descriptions. The reader is referred to Section 2.5.2 and Section 2.5.3, for those details.

Verifier V		**Prover** 1
	Initialisation phase	
	$\xrightarrow{Query}$	Generate random r_1
		$k_1 = PRNG[k_1 \oplus ID_1]$
		$c_1 = c_1 + 1$
If $P_1 \notin$ DB: abort	$\xleftarrow{P_1,r_1,c_1}$	$P_1 = Nun[ID_1,r_1]$
Generate k_{1_v} from c_{1_v}		
$TS = enc$[Timestamp]		
$s_1 = TS \oplus k_{1\_v}$	$\xrightarrow{s_1}$	$TS = s_1 \oplus k_1$
$a^0 = PRF_{k_{1\_v}}[r_1 \|\| TS]$		$a^0 = PRF_{k_1}[r_1 \|\| TS]$
$a^1 = a^0 \oplus k'_{1\_v}$		$a^1 = a^0 \oplus k'_1$
	Distance-bounding and verification phase	
	for $i = 1$ to n	
Pick $c_i \in \{0,1\}$		
Start clock	$\xrightarrow{c_i}$	$r_i = a_i^{c_i}$
Stop clock	$\xleftarrow{r_i}$	
Store r_i and Δ_i		
	end for	
		$b_1 = MAC_{k_1}[TS,ID_1]$
	$\xleftarrow{V,b_1}$	$V = PRF_{k_1}$[transcript]
Verify b_1, V, and all rounds' r_i and Δ_i.		
	$\xrightarrow{Out_v}$	

Figure 3.3: The Distance-Bounding Grouping-Proof Basic 2 (DBGP-B2) protocol for the first prover.

Verifier V		**Prover** j **(for** $j = 2,...,m$**)**
	Initialisation phase	
	$\xrightarrow{b_{j-1}}$	Generate random r_j
		$k_j = PRNG[k_j \oplus ID_j]$
		$c_j = c_j + 1$
If $P_j \notin$ DB: abort	$\xleftarrow{P_1,r_1,c_1}$	$P_j = Nun[ID_j,b_{j-1}]$
Generate k_{j_v} from c_{j_v}		
$a^0 = PRF_{k_{j\_v}}[r_j \| b_{j-1}]$		$a^0 = PRF_{k_j}[r_j \| b_{j-1}]$
$a^1 = a^0 \oplus k'_{j\_v}$		$a^1 = a^0 \oplus k'_j$
	Distance-bounding and verification phase	
	for $i = 1$ to n	
Pick $c_i \in \{0,1\}$		
Start clock	$\xrightarrow{c_i}$	$r_i = a_i^{c_i}$
Stop clock	$\xleftarrow{r_i}$	
Store r_i and Δ_i		
	end for	
		$b_j = MAC_{k_j}[b_{j-1},ID_j]$
	$\xleftarrow{V,b_j}$	$V = PRF_{k_j}$[transcript]
Verify b_j, V, and all rounds' r_i and Δ_i.		
	$\xrightarrow{Out_v}$	
When all provers are done:		
$P_{1,...,m} = (s_1,P_1,b_1,c_1,...,P_j,b_j,c_j,...,P_m,b_m,c_m)$		

Figure 3.4: The Distance-Bounding Grouping-Proof Basic 2 (DBGP-B2) protocol for provers 2 to m.

Details The verifier initiates a session by sending a *Query* to one of the provers, which then will work as the first prover. The prover generates a random value r_1, updates its key k_1 and its counter value c_1, and generates its pseudonym P_1. The prover then sends P_1, r_1 and c_1 to the verifier, which, upon receipt, verifies that the prover is in the database. The verifier then computes the shared key k_{1_v}, an encrypted timestamp TS, and encrypts TS with k_{1_v} into s_1, and sends to the prover. Then, both the verifier and the prover generate the response vectors a^0 and a^1. Following this, the distance-bounding phase begins. In n rounds, the verifier starts a clock and sends a challenge to the prover. When received by the prover, the prover immediately responds with the response r_i from the matching response vector to the verifier. The verifier stops the clock when it receives the response and stores the response r_i and the time delta Δ_i. After the distance-bounding phase, the prover sends its partial proof b_1 and a hash of the transcript, V. The verifier will verify that all rounds' Δ_i were within the allowed Δ_{max}, and that at least τ responses were correct and that the transcript is correct.

The procedure is almost identical for the rest of the provers. But, instead of receiving a query as the first message, they get the previous prover's partial proof. Each prover then generates all the values as the first prover did, and sends this to the verifier. After this, they both compute the response vectors a^0 and a^1 and proceeds in the same way as for the first prover with the distance-bounding phase and verification.

When all provers are finished, the verifier will check that the time taken for the entire proof has not exceeded some pre-defined maximum time. If the time was within the boundaries, the verifier saves all data and partial proofs to its database, from where it later again can be retrieved and verified.

3.3.2 Security analysis

The following section presents the security and privacy analysis of the DBGP-B2 protocol, analysed with respect to the method described in Section 3.1.3. As the protocol is built on the same grouping-proof protocol as DBGP-B1, all of the attack analyses done on grouping-proof attacks in Section 3.2.2 are valid here as well, and are therefore left out of the text. The response vectors are built using sensitive private, but they do not reveal any information that changes the grouping-proof attack analyses done for DBGP-B1.

Distance Fraud

The DBGP-B2 protocol has a challenge $c_i \in \{0,1\}$ and a response $r_i \in \{0,1\}$, where r_i is decided by $r_i = a_i^{c_i}$. As the challenges and responses are the same as in the DBGP-B1 protocol, the distance fraud calculation done in Section 3.2.2 is valid here. It will be the exact same cases, which will exact same probabilities to happen. Therefore, the probability to success with a distance fraud in one round is $Pr[\text{DF}]_i = \frac{3}{4}$, and to pass the distance-bounding phase in all rounds: $Pr[\text{DF}] = (\frac{3}{4})^n$.

Mafia Fraud

As described in Section 2.2.3, there are two ways to get the correct reponse. However, in the DBGP-B2 protocol, the transcript is sent after the distance-bounding phase. This disallows an adversary from passing the protocol if a challenge is guessed wrong and sent to the prover, as that would break the transcript. Because of this, only the first case can be used by an adversary here, as an erroneously guessed challenge would break the proof.

The probability for the first (and only possible) case, guessing the challenge, is simply $\frac{1}{2}$. This is also the total probability of succeeding in one round $Pr[\text{MF}]_i = \frac{1}{2}$, which gives the probability to pass the distance-bounding phase in all rounds: $Pr[\text{MF}] = (\frac{1}{2})^n$.

Terrorist Fraud

The protocol does not protect against terrorist frauds. A malicious prover P^* can give away the information needed to pass the protocol, the vectors a^0 and a^1, without revealing its secret key. The second key, k', can, however, be retrieved by an adversary, but it would be of no help in any future proof sessions, as stated in the analysis in [11].

3.4 The ECC Grouping-Proof Protocol

The following section presents the proposed grouping-proof protocol that has been designed in this thesis, using ECC methods, as well as a security and privacy analysis of it. The protocol will be referred to as ECC-GP.

3.4.1 Description

The protocol is designed with respect to the guidelines to resist attacks against grouping-proofs, presented in Section 2.3.3. As the verifier works a bit differently depending on whether it is the first prover or any of the other provers, the protocol will be presented in two figures. The run of the protocol for the first prover is presented in Figure 3.5, and the run of the protocol for all other provers in the proof, is presented in Figure 3.6.

Verifier V		**Prover** 1
$ts = enc[\text{Timestamp}] \in \mathbb{Z}^*$		
	$\xrightarrow{ts,Y}$	
		Generate random $r_{1,1}$,$r_{1,2} \in \mathbb{Z}^*$
		$R_{1,1} = r_1 P$
		$R_{1,2} = r_2 P$
		$d_{1,1} = xcoord[r_{1,1}Y]$
		$d_{1,2} = xcoord[r_{1,2}Y]$
		$b_1 = ts + x_1 + d_{1,1} + d_{1,2}$
$d'_{1,1} = xcoord[yR_{1,1}]$	$\xleftarrow{b_1,R_{1,1},R_{1,2}}$	
$d'_{1,2} = xcoord[yR_{1,2}]$		
$X'_1 = (b_1 - d'_{1,1} - d'_{1,2})P - tsP$		
Check that $X'_1 \in DB$		

Figure 3.5: The ECC Grouping-Proof (ECC-GP) protocol for the first prover.

Verifier V		**Prover** j **(for** $j = 2,...,m$**)**
	$\xrightarrow{b_{j-1},Y}$	
		Generate random $r_{j,1}, r_{j,2} \in \mathbb{Z}^*$
		$R_{j,1} = r_{j,1}P$
		$R_{j,2} = r_{j,2}P$
		$d_{j,1} = xcoord[r_{j,1}Y]$
		$d_{j,2} = xcoord[r_{j,2}Y]$
		$b_j = b_{j-1} + x_j + d_{j,1} + d_{j,2}$
$d'_{j,1} = xcoord[yR_{j,1}]$	$\xleftarrow{b_j,R_{j,1},R_{j,2}}$	
$d'_{j,2} = xcoord[yR_{j,2}]$		
$X'_j = (b_j - d'_{j,1} - d'_{j,2})P - b_{j-1}P$		
Check that $X'_j \in DB$		
When all provers are done:		
$P_{1,...,m} = (ts,b_1,R_{1,1},R_{1,2},...,b_j,R_{j,1},R_{j,2},...,b_m,R_{m,1},R_{m,2})$		

Figure 3.6: The ECC Grouping-Proof (ECC-GP) protocol for provers 2 to m.

Assumptions It is assumed that the verifier and the provers are setup to use the same domain parameters for the ECC. It is also assumed that the provers' public keys are known to the verifier in advance and stored in a database, else they cannot be verified. For higher security, the public key Y can be initiated to each prover instead of being sent. This would make the prover belong to one verifier though, as the prover must know the verifier's Y to participate in a proof.

Prover 1

The protocol begins by the verifier generating an encrypted timestamp $ts = enc$[Timestamp], which is then sent along with its public key to the first prover. When received by the prover, the prover generates two random numbers $r_{1,1}$ and $r_{1,2}$, two points $R_{1,1} = r_{1,1}P$ and $R_{1,2} = r_{1,2}P$, the x-coordinate of two points $d_{1,1} = xcoord[r_{1,1}Y]$ and $d_{1,2} = xcoord[r_{1,2}Y]$, and its partial proof $b_1 = ts+x_1+d_{1,1}+d_{1,2}$. The prover then sends b_1, $R_{1,1}$ and $R_{1,2}$ back to the verifier. The verifier computes the public key X_1' of the prover and checks if it is in the database. If it is in the database, the prover is verified and the protocol continues. Else, it will abort.

Prover j (for $j = 2,...,m$)

The verifier sends the partial proof of the previous prover, b_{j-1}, and its public key to prover j. Upon receiption, the prover generates two random numbers $r_{j,1}$ and $r_{j,2}$, two points $R_{j,1} = r_{j,1}P$ and $R_{j,2} = r_{j,2}P$, the x-coordinate of two points $d_{j,1} = xcoord[r_{j,1}Y]$ and $d_{j,2} = xcoord[r_{j,2}Y]$, and its partial proof $b_j = b_{j-1} + x_j + d_{j,1} + d_{j,2}$. The prover then sends b_j, $R_{j,1}$ and $R_{j,2}$ back to the verifier. The verifier computes the public key X_j' of the prover and checks if it is in the database. If it is in the database, the prover is verified and the protocol continues. Else, it will abort.

Proof

When all provers are done, the verifier checks so that $current\ time - timestamp$ is within some pre-defined maximum time. If it is, the verifier stores the proof $P_{1,...,m} = (ts, b_1, R_{1,1},R_{1,2}, ..., b_j, R_{j,1},R_{j,2}, ..., b_m, R_{m,1},R_{m,2})$ to its database. This proof can later be validated again by someone who knows the secret key, y, of the verifier.

Verification

To verify that the prover is in possession of the secret key, and is who it claims to be, some mathematical computations can be done on the proof. These computations allows for the verifier to verify the prover, without being able to recover the private key. It can be seen that:

$$b_jP = b_{j-1}P + xP + d_{j,1}P + d_{j,2}P \iff X_j = b_jP - d_{j,1}P - d_{j,2}P - b_{j-1}P$$

This allows the verifier to calculate the public key in the following way (for the first prover: replace b_{j-1} with ts):

$$X_j = b_j P - d_{j,1} P - d_{j,2} P - b_{j-1} P = (b_j - d_{j,1} - d_{j,2}) P - b_{j-1} P$$

3.4.2 Security and Privacy Analysis

This section presents a security and privacy analysis of the ECC-GP protocol. The analysis is done using the methods described in Section 3.1.3.

Replay, Subset Replay, and Multiple Impersonation Attacks

The protocol follows the guidelines to resist these attacks. Each session is started by the verifier generating an encrypted timestamp, which then is used by the first prover when it generates its partial proof. Each succeeding prover then builds its partial proof using the partial proof from the preceding prover, which makes all proofs chained all the way back to the encrypted timestamp. This makes all provers' proofs dependent on the timestamp, as well as on preceding provers' proofs, and thus all the proofs are dependent on this unique session.

Anonymity Attacks

The prover never sends any ID nor pseudonym to the verifier at all. Instead, a prover is identified by its public key X_j, which is calculated from its partial proof b_j. A verification process that can only be performed by the verifier, as it is the only one that can make use of the partial proof from the prover.

To be able to calculate the prover's public key, X_j, and thus be able to identify the prover, $x_j P$ must be extracted from b_j. To be able to get the point $X_j = x_j P$ from $b_j P$, both $d_{j,1}$ and $d_{j,2}$ must be known, so that they can be deducted from the proof. Both of these values are calculated by $d_{j,x} = xcoord[r_{j,x} Y]$. As the points $R_{j,x} = r_{j,x} P$ are sent to the verifier, which is in possession of the private key y, the verifier is able to calculate $d_{j,x}$ also, since $r_{j,x} Y = y R_{j,x}$. Since only the verifier knows the private key y, this disallows for any adversary to calculate $d_{j,x}$, hence no one but the verifier can identify the prover.

Traceability Attacks

The way b_j is calculated makes the protocol resistant to traceability attacks. All data but the key x_j is different in each sessions, thus making b_j different for each prover in every run of the protocol.

Forged Proofs

To impersonate a prover and be able to create a partial proof using that prover's data, an adversary A would need to be able to calculate a valid proof b_j. In order to do this, A would need to get possession of the prover's private key x_j, as everything else is randomly selected and dependent on the current session. However, getting in possession of x_j is impossible without physical access to the prover.

Forward Security

This protocol provides forward security. If an adversary would get physical access to some prover j, it could get access to the prover's private and public keys x_j and X_j. In the following proof, it is also assumed that the adversary has the data from the proof: b_{j-1}, b_j, $R_{1,1}$, and $R_{1,2}$.

To see if the prover j was in this proof b_j, the adversary must re-calculate the proof and see if it matches. Using the proof calculation $b_j = b_{j-1} + x_j + d_{j,1} + d_{j,2}$, it follows clearly that it is impossible to reverse the calculation, as both $d_{j,1}$ and $d_{j,2}$ are unknown to the adversary.

As $R_{j,1}$ and $R_{j,2}$ are points, it may look possible to use point multiplication to reverse the calculation. This, however, will not work either, as it is the x-coordinate of the points $r_{j,1}Y$ and $r_{j,2}Y$ that needs to be used, and not the points $R_{j,1}$ and $R_{j,2}$. To get the x-coordinates of the correct points, the private key y of the prover must be known and multiplied with the points $R_{j,1}$ and $R_{j,2}$.

Race Conditions

The protocol is resistant to race conditions. The provers are static and are only called once in each proof session. To create their proof, they receive some information, process this information and immediately send a response when they are done.

Denial of Proof (m-DoP)

Any m-DoP attack against the protocol would fail, since the verifier is in online mode and each prover is verified immediately after the verifier gets the prover's partial proof.

Denial of Service (DoS)

The provers are static and store no data between sessions, nor do they update any data before or after a session. Thus, there is nothing in a prover that can be desynchronised.

3.4.3 Comparison With Other ECC Grouping-Proof Protocols

This section will give a brief discussion that compares the ECC-GP protocol to the other two ECC grouping-proof protocols seen in the literature, which are presented in [29] and [30].

Discussion The analysis in Section 3.4.2 shows that the ECC-GP protocol follows the guidelines for each of the known attacks, and that it is not vulnerable to any of the security or privacy attacks.

The protocol proposed by Batina et al. in [29] has been shown to be insecure against two attacks by Hermans and Peeters in [30].

The other protocol, that looked more secure, was proposed by Hermans and Peeters in [30]. The protocol does not have any dependencies between the provers in the protocol, and does thus not follow the guidelines that have been setup to tolerate the subset replay and the multiple impersonation attacks (see Section 2.3.3). Not following the guidelines does not automatically make the protocol vulnerable to the attacks, and the protocol needs to be further reviewed. However, Peris-Lopez et al. clearly points out the importance of following the guidelines, as they write that they "... should be followed by protocol designers to preclude past errors" [25].

On the contrary, the protocol proposed in [30] is vulnerable to race conditions. The protocol begins with the provers generating two random numbers r_1 and r_2 each, and then send these as points to the verifier. The verifier then sends an "exam" e to the provers. Upon reception, the provers create a proof $s = e*x+r_1+r_2$ and sends back to the verifier. If two verifiers contact a prover at the same time, the prover will generate two sets of random numbers and not know to which of the verifiers they belong, and the proof may thus be corrupted.

Conclusion The conclusion drawn from this is that the ECC-GP protocol is more secure than the other two protocols seen in the literature. The ECC-GP protocol has been proven to be secure against all the known security and privacy attacks, something the other two protocols have been proven not to be.

3.5 The Distance-Bounding Grouping-Proof Secure Protocol

The following section will present the Distance-Bounding Grouping-Proof Secure protocol (DBGP-S), as well as a security and privacy analysis of it. The protocol is built by combining the distance-bounding protocol HPO, presented in Section 2.5.4, and the grouping-proof protocol proposed in this thesis, the ECC-GP protocol, presented in Section 3.4.

3.5.1 Description

As the verifier works differently for the first prover and the rest of the provers, the presentation of the protocol is split into two figures. The run of the protocol for the first prover is presented in Figure 3.7 and the run of the protocol for all other provers in the proof is presented in Figure 3.8. Technical descriptions that have not changed from the original protocols are left out of the text. For details about these, the reader is referred to Section 2.5.4 and Section 3.4.

Verifier V		**Prover** 1
	Initialisation phase	
$ts = enc[\text{Timestamp}]$		
Generate random r_v		
$R_v = r_v P$	$\xrightarrow{ts, R_v, Y}$	Generate random $r_{1,1}, r_{1,2}$
		$R_{1,1} = r_{1,1}P$
	$\xleftarrow{R_{1,1}, R_{1,2}}$	$R_{1,2} = r_{1,2}P$
$a^0 \vert\vert a^1 = [\text{xcoord}(r_v R_{1,1})]_{2n}$		$a^0 \vert\vert a^1 = [\text{xcoord}(r_{1,1} R_v)]_{2n}$
$c = [e]_n,\ e \in \mathbb{Z}^*$		
	Distance-bounding and verification phase	
	for $i = 1$ to n	
Start clock	$\xrightarrow{c_i}$	$r_i = a_i^{c_i}$
Stop clock	$\xleftarrow{r_i}$	
Store r_i and Δ_i		
	end for	
Check all r_i and Δ_i		
	$\xrightarrow{e}$	Verify $c_{n-1} \vert\vert .. \vert\vert c_1 \vert\vert c_0 == [e]_n$
		$d_{1,2} = xcoord[r_{1,1}Y]$
		$d_{1,2} = xcoord[r_{1,2}Y]$
$d'_{1,1} = xcoord[yR_{1,1}]$	$\xleftarrow{b_1}$	$b_1 = ts + x_1 + e + d_{1,1} + d_{1,2}$
$d'_{1,2} = xcoord[yR_{1,2}]$		
$X'_1 = (b_1 - d'_{1,1} - d'_{1,2})P - tsP - eP$		
Verify $X'_1 \in$ DB	$\xrightarrow{Out_V}$	

Figure 3.7: The Distance-Bounding Grouping-Proof Secure (DBGP-S) protocol for the first prover.

Verifier V		**Prover** j **(for** $j = 2,...,m$**)**
	Initialisation phase	
Generate random r_v		
$R_v = r_v P$	$\xrightarrow{b_{j-1},R_v,Y}$	Generate random $r_{j,1}, r_{j,2}$
		$R_{j,1} = r_{j,1}P$
	$\xleftarrow{R_{j,1},R_{j,2}}$	$R_{j,2} = r_{j,2}P$
$a^0\|\|a^1 = [\text{xcoord}(r_v R_{j,1})]_{2n}$		$a^0\|\|a^1 = [\text{xcoord}(r_{j,1} R_v)]_{2n}$
$c = [e]_n,\ e \in \mathbb{Z}^*$		
	Distance-bounding and verification phase	
	for $i = 1$ to n	
Start clock	$\xrightarrow{c_i}$	$r_i = a_i^{c_i}$
Stop clock	$\xleftarrow{r_i}$	
Store r_i and Δ_i		
	end for	
Check all r_i and Δ_i		
	$\xrightarrow{e}$	Verify $c_{n-1}\|\|..\|\|c_1\|\|c_0 == [e]_n$
		$d_{1,2} = xcoord[r_{j,1}Y]$
		$d_{1,2} = xcoord[r_{j,2}Y]$
$d'_{j,1} = xcoord[yR_{j,1}]$	$\xleftarrow{b_j}$	$b_j = b_{j-1} + x_1 + e + d_{j,1} + d_{j,2}$
$d'_{j,2} = xcoord[yR_{j,2}]$		
$X'_j = (b_j - d'_{j,1} - d'_{j,2})P - b_{j-1}P - eP$		
Verify $X'_j \in$ DB		
	$\xrightarrow{Out_V}$	
When all provers are done:		
$P = (ts,b_1,R_{1,1},R_{1,2},...,b_j,R_{j,1},R_{j,2},...,b_m,R_{m,1},R_{m,2})$		

Figure 3.8: The Distance-Bounding Grouping-Proof Secure (DBGP-S) protocol for provers 2 to m.

Details The protocol begins by the verifier generating an encrypted timestamp ts, a random number r_V, and a point $R_V = r_V P$. The point R_V, the timestamp ts, and the verifier's public key Y is then sent to some prover, noted as the first prover. Upon reception, the prover will generate two random values $r_{1,1}$ and $r_{1,2}$, two points $R_{1,1} = r_{1,1}P$ and $R_{1,2} = r_{1,2}P$, and then send these two points to the verifier. Following that, both the verifier and the prover will calculate the response vectors a^0 and a^1, by taking the first $2n$ bits from the x-coordinate of the point $r_v R_{1,1} = r_{1,1}R_V = r_v r_{1,1}P$. The verifier will also compute the challenge vector, by taking the first n bits from some random number e. Following that comes the the distance-bounding phase. For n rounds, the verifier will start a clock and send the i-th bit of the challenge vector c to the verifier. The prover will immediately respond with the i-th bit of the response vector that matches challenge: $r_i = a_i^{c_i}$.

When the verifier receives the response, it will stop the clock and store the response r_i as well as the time delta Δ_i of the round. When the distance-bounding phase is over, the verifier verifies that r_i for each round was correct, and that each round's Δ_i was within the allowed Δ_{max}. The verifier then sends the random number e to the prover. The prover then verifies e against the challenges received and, if it was carried out with no problem, creates the partial proof b_1, which is sent as response to the verifier. The verifier then verifies the proof by calculating the public key of the prover and checks if the calculated public key is in its database. This calculation also assures that e has not been tampered with.

The run of the protocol for all the other provers proceeds in the same way, with one difference. Instead of generating an encrypted timestamp and send to a prover, the verifier will send the previous prover's partial proof. The rest of the protocol runs exactly as described for the first prover.

When all provers are done, the verifier will check that the entire protocol was run within some maximum allowed time delta, and then store all data and partial proofs to its database, from where it later may be retrieved and verified again.

3.5.2 Security and Privacy Analysis

The following section presents the security and privacy analysis of the DBGP-S protocol, analysed using the methods described in Section 3.1.3.

Distance Fraud

The DBGP-S protocol has the challenge $c_i \in \{0,1\}$ and the response $r_i \in \{0,1\}$, where the response is decided by $r_i = a_i^{c_i}$. As in the previous distance fraud analyses, there are two cases:

1. The best case for some malicious prover P^* is when $a_i^0 = a_i^1$. The response is the same whether c_i is 0 or 1. The probability to be successful in this case is 100% and the probability for this to happen is $\frac{1}{2}$. Thus, the total probability for having this case and being successful is: $Pr[\text{case 1}] = 1 * \frac{1}{2}$.

2. The other case, when $a_i^0 \neq a_i^1$, happens with probability $1 - \frac{1}{2} = \frac{1}{2}$. This time, P^* needs to guess the response. As the response can take two values, the probability of P^* making a correct guess is $\frac{1}{2}$. The probability for this case to happen and that P^* makes a correct guess is then: $Pr[\text{case 2}] = \frac{1}{2} * \frac{1}{2} = \frac{1}{4}$.

This gives a total probability to pass one round: $Pr[\text{DF}]_i = Pr[\text{case 1}] + Pr[\text{case 2}] = \frac{1}{2} + \frac{1}{4} = \frac{3}{4}$, and to pass the distance-bounding phase in all rounds: $Pr[\text{DF}] = (\frac{3}{4})^n$.

Mafia Fraud

From Section 2.2.3, we know that there are two cases in which an adversary can succeed with a mafia fraud. However, the first case of guessing the challenge c_i and send in advance to P cannot be used, as the challenges are verified after the distance-bounding phase. If the adversary would send a challenge to the prover that is different from what the verifier sent, this will be seen by the prover after it has received e and calculated the challenge vector, and compares against the challenges received. The adversary cannot change e either to match what was maliciously sent, as the received e is used to create the prover's partial proof. If it has been changed, the verifier will not be able to verify the prover. The adversary is thus stuck to only being able to guess the responses. With two possible response values, the probability of making a correct guess is $\frac{1}{2}$.

The total probability for succeeding in one round is then $Pr[\text{MF}]_i = \frac{1}{2}$, and to pass the distance-bounding phase in all rounds: $Pr[\text{MF}] = (\frac{1}{2})^n$.

Terrorist Fraud

This protocol does not protect against terrorist frauds. A malicious prover could simply give away the vectors a^0 and a^1 to an adversary, without revealing its secret key.

The values are obtained from $a^0||a^1 = [\text{xcoord}(r_{j,1}r_vP)]_{2n}$, i.e. $2n$ bits of the x-coordinate of the point $r_{j,1}r_vP$. This will not give the adversary any future help at all, since both $r_{j,1}$ and r_v are two randomly chosen values. These were chosen between this verifier and this prover for this session. In the next run of the protocol, they will not be the same.

Grouping-Proof Attacks

The security and privacy analysis that was made in Section 3.4.2 is still valid for the attacks against this protocol, except the *race condition* attack. The race condition attack, however, can not be fully protected against when the protocol has distance-bounding properties, as the response vectors for the distance-bounding phase must be calculated in advance, and be unique for the prover and the verifier for that session only.

For the replay, subset replay, multiple impersonation, m-DoP, and DoS attacks, it follows clearly that none of the new parameters added affects them. But for the other attacks, it is not as obvious. Therefore, descriptions of how the added values and calculations (a^0, a^1, and e) do not affect the security against the other grouping-proof attacks are described.

Implication of the response vectors a^0 and a^1

The one main thing that can be seen to impact the security in some way is the response vectors a^0 and a^1, as they are calculated using some data that is also used when creating a prover's partial proof. However, the way a^0 and a^1 are calculated, they do not reveal any information about the variables involved, nor is the variable r_v used in any other point than this. The response vectors are calculated by taking $2n$ bits of the x-coordinate of the point $r_{j,1}r_vP$. As the discrete logarithm assumption states, there is no way to reverse the values of $r_{j,1}$ or r_v.

The use of only $2n$ bits of the point's x-coordinate makes it even more useless for an adversary, as it would first need to reverse the function to get the point, which then, as shown, would be of no use anyway.

Implications of adding e to the partial proof

Adding the e vector to the partial proofs does not change anything that would invalidate the security and privacy analysis that was done for the ECC-GP in Section 3.4.2. As the e variable is added by simple addition, it does in no way help an adversary to extract $d_{j,1}$ or $d_{j,2}$, nor helps in being able to use $R_{j,1}$ and $R_{j,2}$ for verification.

If an adversary would be able to forge the value of e sent to the prover, there is no value it can be set to that would give any advantage to the adversary. The values $d_{j,1}$ and $d_{j,2}$ still makes the partial proof non-static and totally random for this session. Thus, setting e to some value used in a previous session would not work, as the values $d_{j,1}$ and $d_{j,2}$ were different in that session.

With the above description of the impact of the value e to the partial proof, it follows that the protocol still is secure against anonymity and traceability attacks, gives no information that makes it possible for an adversary to create forged proof, and still provides forward security.

3.6 Protocols Comparison

Table 3.1 shows an overview of the protocols proposed in this thesis. It shows the success probabilities for the distance-bounding attacks and whether the protocols are secure against the grouping-proof attacks or not. As the ECC-GP is a grouping-proof protocol, the success probabilities of the distance-bounding attacks are not applicable.

Protocol	ECC-GP	DBGP-B1	DBGP-B2	DBGP-S
Distance Fraud	-	$(\frac{3}{4})^n$	$(\frac{3}{4})^n$	$(\frac{3}{4})^n$
Mafia Fraud	-	$(\frac{3}{4})^n$	$(\frac{1}{2})^n$	$(\frac{1}{2})^n$
Terrorist Fraud	-	1	1	1
Replay attacks	Y	Y	Y	Y
Subset attacks	Y	Y	Y	Y
Multiple impersonation attacks	Y	Y	Y	Y
Anonymity attacks	Y	Y	Y	Y
Traceability attacks	Y	Y	Y	Y
Forged proofs	Y	Y	Y	Y
Forward security	Y	N	N	Y
Race conditions	Y	N	N	N
m-DoP attacks	Y	Y	Y	Y
DoS attacks	Y	Y	Y	Y

Table 3.1: Comparsion of the protocols designed in this thesis. Shows the distance-bounding attack success probabilities and whether or not they are secure against the grouping-proof attacks.

4

Simulation and Test Results

This chapter will present the simulation framework that has been built to simulate the protocols, and its implementation details. Following that comes a presentation of the results from the simulation of the protocols, which have been tested in efficiency, false rejections, and security.

4.1 Simulation Implementation

In order to simulate the protocols, a simulation framework was developed. The framework does not provide a genuine RFID environment, but it enables to test whether a protocol can run, how it is affected by noise and different tolerated noise levels, and how it withstands distance fraud and mafia fraud attacks. This section will first give an overview of how the implementation was built. Following that is a more detailed description of some, for the test results, more crucial parts.

4.1.1 Framework

The simulation framework is built so that it provides the basic functionality needed to simulate a reader and tags. To use the framework and simulate a protocol, a user creates classes that extends the framework's basic classes, and adds the specific tag and reader code. This extended code is then what is run by the simulation framework. This allows for a user implementing a protocol to only care about the protocol-specific code, and not need to care about how readers and tags are set up and how they connect, as it is provided by the framework.

Framework Class Details The module `simulator` is the base of the simulation program. This module has three classes: `Reader`, `Tag` and `Simulator`. The `Reader` class contains all the basic functionality needed for a reader to work, which is: sending and receiving data from a tag, check for tags, connect to a tag, run the extended class's code when connected to a tag, and call a function to print a summary when done. The `Tag` class is much simpler, and contains methods for: sending and receiving data, running the extended class's code when connected to a reader, and terminating the tag. The `Simulator` class is the one containing all simulation information. The constructor of the class needs information about the extending classes that implement the protocols and the settings under which the simulation will be run. This class contains the `run` method, which starts the simulation. When the method is invoked, the simulator will create a reader object, and objects for all tags, and have them run as own processes.

4.1.2 Noise

To simulate the noise, a special code is run on the reader process when sending and receiving data. This code is only run when executing the distance-bounding phase, as the other phases are considered to have error correcting code. The pseudo code in Listing 4.1 shows the implementation of noise when the reader sends a challenge to the tag, and the pseudo code in Listing 4.2 when a response is received, i.e. when the tag sends to the reader.

Listing 4.1: Pseudo code for noise when sending data.

```
Send(challenge):
  if  use_noise and db_phase_runs:
    rand := Random integer between 0 - 10^6
    if rand <= Reader_Noise:
      challenge := (challenge + 1) % 2

    Do_Send(challenge)
```

Listing 4.2: Pseudo code for noise when receiving data.

```
Receive(response):
  if  use_noise and db_phase_runs:
    rand := Random integer between 0 - 10^4
    if rand <= Tag_Noise:
      response := (response + 1) % 2

    Do_Receive(challenge)
```

4.1.3 Attacks

To simulate the distance fraud and the mafia fraud attacks, special methods for each of them were created. When simulating the attacks, the corresponding method is run on the tag, instead of the regular distance-bounding code. The pseudo code for the distance fraud attack is seen in Listing 4.3.

Listing 4.3: Pseudo code for the distance fraud attack.

```
Distance_Fraud(round):
  if a0[round] == a1[round]:
    response := a0
  else:
    response := Random integer between 0 - 1

    Send(response)
```

To simulate the behaviour of an adversary executing a mafia fraud attack, when it is possible to send a challenge in advance to the tag, the pseudo code in Listing 4.4 is used. Line 2 represents the adversary guessing a challenge and “sending to the tag”. Then, line 3 is where the adversary “receives” the response from the tag.

Listing 4.4: Pseudo code for the mafia fraud attack when querying the tag.

```
 1 Mafia_Fraud(round, challenge):
 2   guessed_challenge :=  Random integer between 0 - 1
 3   tag_response := a[guessed_challenge][round]
 4
 5   if guessed_challenge == challenge:
 6     response := tag_response
 7   else:
 8     response :=  Random integer between 0 - 1
 9
10     Send(response)
```

Listing 4.5: Pseudo code for the mafia fraud attack when not querying the tag.

```
Mafia_Fraud(round, challenge):
  response := Random integer between 0 - 1

  Send(response)
```

4.2 Test Setup

This section will present which protocols that have been tested and what test system that has been used.

4.2.1 Tested Protocols

Only the two protocols DBGP-B1 and DBGP-S have been implemented and tested. The reason why only these two protocols were implemented is that they cover all interesting test cases:

- **Efficiency** This tests aims to see the efficiency difference between the basic protocol, designed for regular RFID tags, and the ECC based protocol, designed for more powerful RFID tags. Therefore, testing only DBGP-B1 and DBGP-S is sufficient enough to see how using ECC affects the efficiency.
- **False Rejections** This test can be run on any of the protocols and does not differ in different protocol implementations. The only thing that concerns this test is the communication channel noise, which is handled the same for all implementations.
- **Security** An adversary has a success probability of $\frac{3}{4}$ in both a distance fraud and a mafia fraud attack against the DBGP-B1 protocol. Against the DBGP-S protocol, an adversary has a success probability of $\frac{3}{4}$ in a distance fraud attack, and a success probability of $\frac{1}{2}$ in a mafia fraud attack. The success probabilities against the DBGP-S protocol is the same for the DBGP-B2 protocol. As the attacks would be created in the same way for the DBGP-S and the DBGP-B2 protocols, it is sufficient enough to only test them against the DBGP-S protocol.

  Testing the attacks against both DBGP-B1 and DBGP-S thus covers all success probabilities against all protocols designed in this thesis, although the protocols are different.

4.2.2 Test System

The crucial system components and program versions of the test system on which all efficiency tests have been run can be seen in Table 4.1.

CPU	Intel Core i7-2700k, clocked to 4.6GHz.
Ram	16GB.
Hard Drive	SSD.
Operating System	Arch Linux (x64).
Kernel version:	3.14.1-1-ARCH.
Python version	3.4.0.
sqlite3 version	3.8.4.3 2014-04-03 16:53:12.

Table 4.1: Test system used for efficiency tests.

4.3 Test Results

In this section, the simulation results of the DBGP-S and DBGP-B1 protocols, using the simulation framework created, will be presented. The test data for all results presented in this section can be found in Appendix A.

4.3.1 Efficiency

To test the efficiency of the protocols, the implementations of them have been run with different number of tags and distance-bounding rounds, 1,000 times for each setup. For each run, the time it took to run the simulation program has been measured and the average value has been calculated over all 1,000 runs. The test system used for all tests can be seen in Section 4.2.2.

Both the DBGP-B1 and the DBGP-S protocols were tested when using 3, 10, 25, and 50 tags, to see how the protocols are affected by an increasing number of tags. For each number of tags, the protocols were tested with both 50 and 100 distance-bounding rounds, to see how much the number of rounds affects the time it takes to run the protocol. The results can be seen in Figure 4.1. The x-axis is the number of tags used, and the y-axis is the time it took to execute the simulation program, measured in seconds.

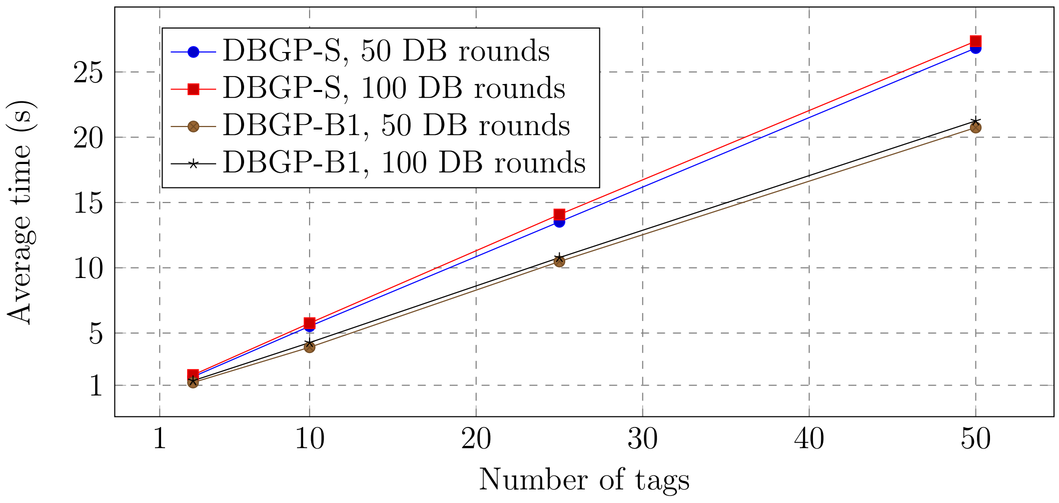


Figure 4.1: Simulation results of protocol efficiency. Total time it takes to run the two protocols with x tags, when using 50 or 100 distance-bounding rounds.

The numbers (exact number can be seen in Appendix A.1) shows that going from 50 to 100 distance-bounding rounds, gives a $2\% - 8\%$ run time increase for the DGBP-S protocol, and a $2\% - 12\%$ run time increase for the DBGP-B1 protocol. The relative time difference between 50 and 100 distance-bounding rounds gets lower as the number of tags increases, for both protocols.

Comparing the DBGP-S protocol with the DBGP-B1 protocol, it can be seen that for 50 distance-bounding rounds the DBGP-B1 protocol is $29\% - 41\%$ faster. With 100 distance-bounding rounds, the DBGP-B1 protocol is $29\% - 35\%$ faster. As the number of tags increases, the overhead of starting up the simulation framework gets lower. It is therefore interesting to see that both protocols are around $29\% - 30\%$ faster for both 50 and 100 distance-bounding rounds when using 25 and 50 tags.

4.3.2 False Rejections

When there is noise in the communication channel, there is always a possibility that a prover will be a victim of a false rejection, i.e. denied even though it is legit. This might happen even though the protocol is set up to tolerate noise to a certain level. Figure 4.2 shows the results when testing for false rejections, on a protocol that is set up to tolerate 10% noise. The x-axis is the number of rounds and the y-axis is the probability that a legit user is falsely rejected. In tests that resulted in no false rejections, the result is excluded from the figure. That is why the brown and the red lines stops at 20 and 80 distance-bounding rounds, respectively.

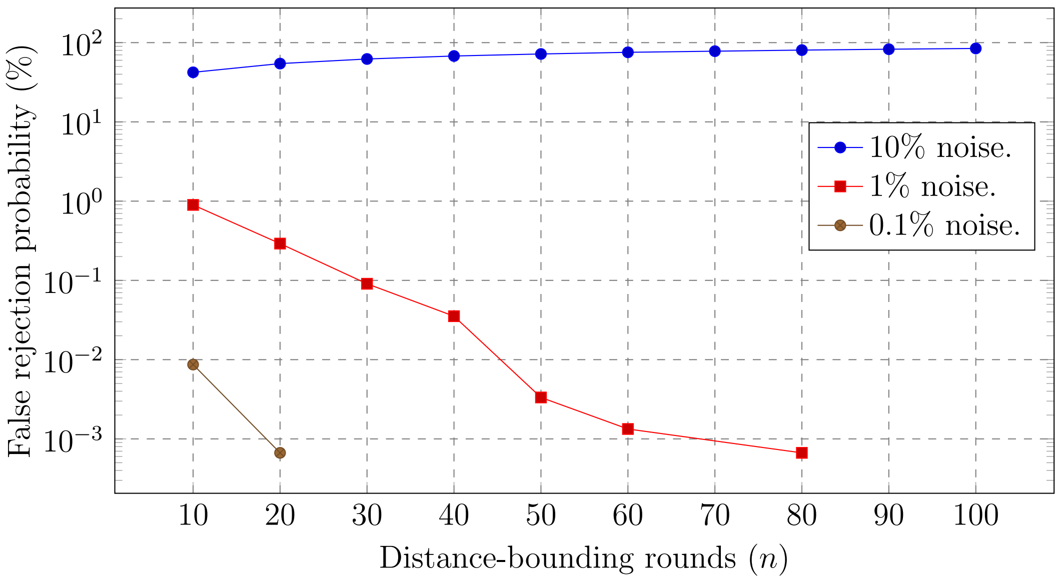


Figure 4.2: Simulation that shows how the probability of a legitimate prover to be falsely rejected is affected by noise in the communication channel, when protocol is set up to tolerate 10% noise.

With communication noises lower than 0.1%, no test resulted in a false rejection. As seen in the figure, with 0.1% noise, no legitimate prover got rejected when using more than 20 distance-bounding rounds. For 1% noise, no legitimate user got rejected when using more than 80 distance-bounding rounds.

With 10% communication noise, the false rejection rate starts at 42% when using 10 distance-bounding rounds, and increases up to 84% when using 100 distance-bounding rounds. The blue line, which shows when there is 10% noise, has an opposite behaviour compared to the other lines. As the number of distance-bounding rounds increases, the false rejection rate also increases in this case. For 1% noise and lower, the false rejection rate decreases. An important note here, is that a protocol tolerating 10% noise is set up to accept a prover if 90% of the distance-bounding rounds were correct. At the same time, having 10% noise in the communication channel means that there is a 10% chance that a bit is being modified, *both* when being sent and received by the reader.

4.3.3 Security

In order to determine how secure the protocols are, simulated distance fraud and mafia fraud attacks have been tested against them. The results of simulating these attacks show to what success probability an adversary or a malicious tag can pass the protocol. This success probability is what defines how secure they are.

Both protocols have been set up to tolerate a noise level of 10%. They have then been tested against the two attacks with different levels of noise in the communication channel, to see how the security is affected by this. The noise on the channel has in all tests ranged from 0% to 10%.

Distance Fraud

Figure 4.3 shows the results when testing the distance fraud attack. As the results were nearly the same when testing for distance fraud against the DBGP-S and DBGP-B1 protocols, only the results from the DBGP-S are shown (the results for DBGP-B1 are available in Appendix A.3.1). There is no value for the 10% noise and 100 distance-bounding rounds, as none of the attack attempts succeeded. The black, theoretical line without noise, is calculated by using Equation 2.1 as follows $B(n, 0.9n, \frac{3}{4})$.

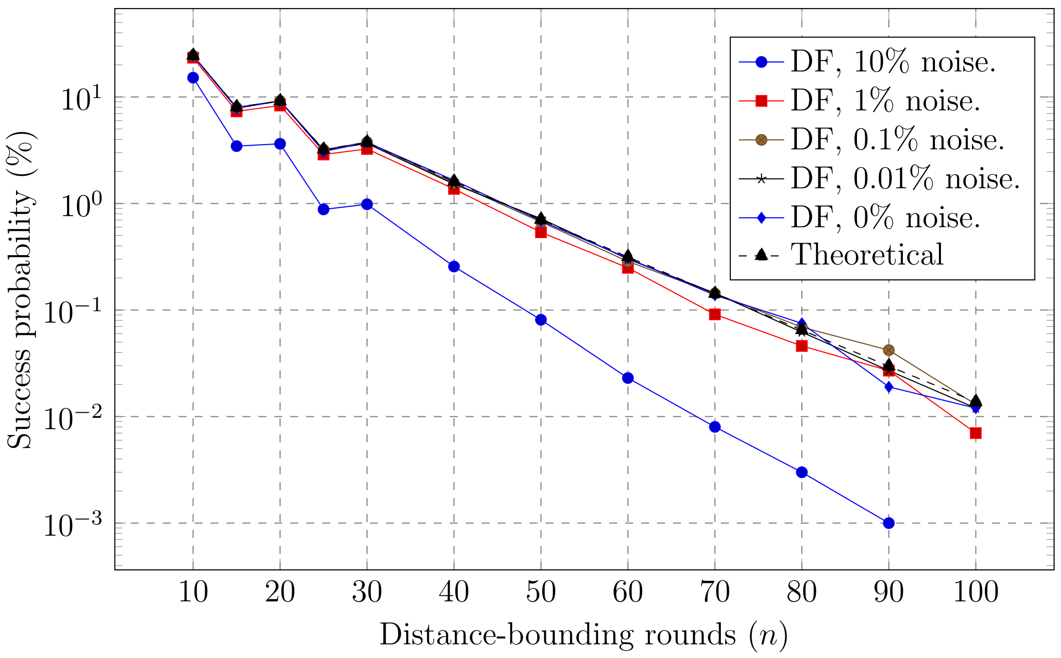


Figure 4.3: Simulation of how the success probability of a distance fraud attack against the DBGP-S protocol, set-up to tolerate 10% noise, is affected by different levels of noise in the communication channel.

From the simulation results, it is easy to see that with higher noise on the channel, the success probability for a malicious tag decreases. The blue line, that shows the probability when having 10% noise on the channel, is far below the black, theoretical line. Also the red line, that shows the probability when having 1% noise on the channel, is a bit below the theoretical line. As the noise decreases, the success probability increases. This is seen by all the lines getting closer to the theoretical line, as the noise gets lower.

Mafia Fraud

Figure 4.4 shows the results when testing mafia fraud attacks against the protocols. With 10% noise in the communication channel, there were no successful attempts when using 90 or 100 distance-bounding rounds. For the DBGP-S protocol though, regardless of the noise, no successful attempts were made when using more than 30 distance-bounding rounds. All tests that resulted in no successful attempt have been excluded from the figure, which is why some lines stops at earlier than 100 distance-bounding rounds.

The black, theoretical lines without noise, are calculated by using Equation 2.1 as follows $B(n,\, 0.9n,\, q)$. The solid black line, representing the theoretical success probabilities against DBGP-S, was calculated with $q = \frac{1}{2}$, and the dashed black line, representing the theoretical success probabilities against DBGP-B1, was calculated with $q = \frac{3}{4}$.

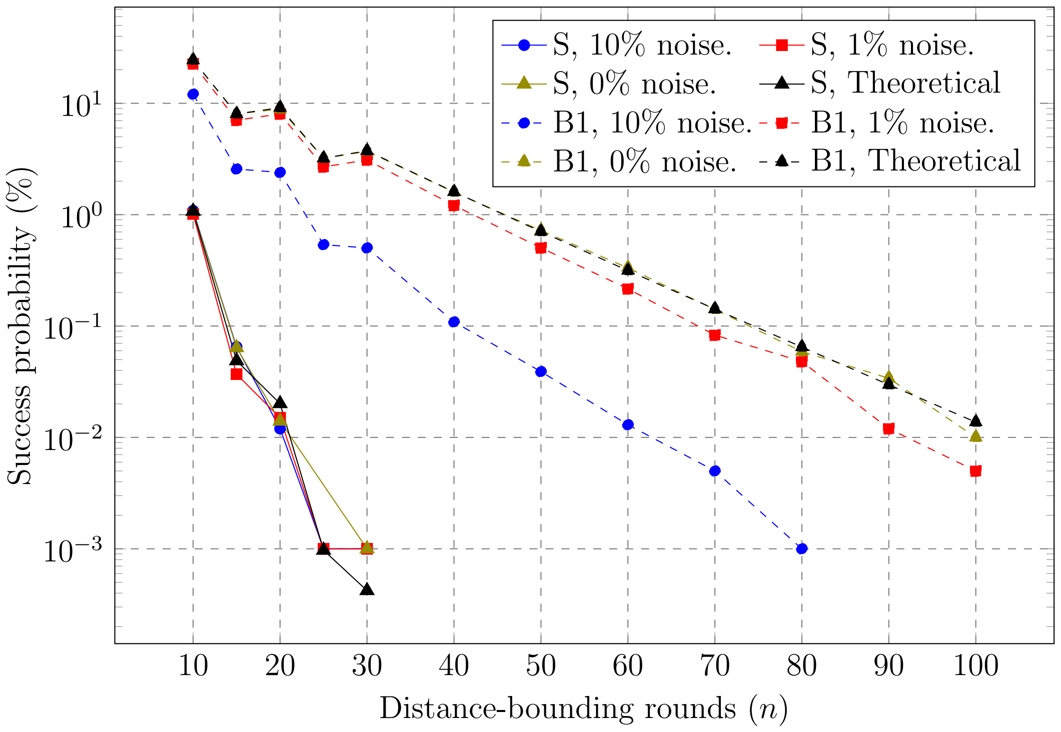


Figure 4.4: Simulation of how the success probability of a mafia fraud attack against the DBGP-S and the DBGP-B1 protocols, set-up to tolerate 10% noise, is affected by different levels of noise in the communication channel.

It is easy to see that the DBGP-S protocol is more secure than the DBGP-B1 protocol against the mafia fraud attack. This was expected though, as the DBGP-S protocol provides better security by its design (see comparison in Section 3.6).

The success probability for the DBGP-S protocol, shown in solid lines, can be interpreted as not being affected by noise in the communication channel. The lines representing the different communication channel noise settings all follow the theoretical, black line very closely. The DBGP-B1 protocol, shown in dashed lines, on the other hand, is notably affected by the noise. It is easy to see as the blue line, showing the success probability with 10% noise in the communication channel, is located far below the other lines for this protocol. The success probability of the attack can be interpreted as being lower the more noise there is on the channel, just as in the case of the distance fraud attack. It is easy to interpret it that way, as the lines gets closer to the theoretical line, the less noise there is.

Comparing the security of the two protocols against each other shows a clear advantage for the DBGP-S protocol. The success probability also decreases faster for the DBGP-S protocol than the DBGP-B1 protocol, something that the numbers, seen in Appendix A.3.2, clearly show. With 1% noise in the communication channel, 1,015 of 100,000 attempts were successful against the DGBP-S protocol, when using 10 distance-bounding rounds. For the DBGP-B1 protocol, the number of successful attempts were 22,636 at the same settings. This gives success probabilities of 1.015% and 22.64%, for the DBGP-S and the DBGP-B1 protocols, respectively. A 22 times higher rate for the DBGP-B1, compared to the DBGP-S. With the same noise setting, at 20 distance-bounding rounds, only 15 of 100,000 attempts against the DBGP-S protocol were successful. That gives a success rate of 0.015%. For the DBGP-B1 protocol, using 20 distance-bounding rounds resulted in 8,026 successful attempts, which gives a success rate of 8.026%. That is 535 times more than for the DBGP-S protocol.

5

Discussion

In this chapter, a discussion about the method, the results, and future work will be presented.

5.1 Method

Finding and reading published articles about the two types of protocols that have been used in this thesis have had both advantages and disadvantages. The advantages are that many good researchers have been studying this area and published their results, which others then can use. As I have seen there are new protocols presented all the time, as they seem to find vulnerabilities in previous ones. Having the opportunity to find the latest and best, and use them in this thesis have been of great importance. The disadvantages of using this research method is that not all articles that are published seem to be of high quality. It can be hard to distinguish which articles that are the most relevant, which in turn can lead to using something that is not the latest state of the art research.

Creating a simulation framework to simulate protocols on RFID tags and an RFID reader has been an adequate way to tell whether the protocols can be implemented. It has also been of good use to realistically test how the protocols stand the implemented attacks, how different levels of noise affects them, and how different techniques affects the efficiency of the protocols. It is hard though, to tell whether the protocols can be realised on real RFID tags and readers. Even if they theoretically can be implemented, it is hard to tell whether they will be efficient enough when having the constraints that a real RFID tag has. To get an answer to this, more constraints need to be added to the simulation framework, e.g., used hardware, communication speeds, noise in different environments.

5.2 Results

This section discusses the results of this thesis. It starts with a discussion about the protocols that have been created, which is followed by a discussion of the simulation results.

5.2.1 Designed Protocols

In this thesis, a new type of protocol has been proposed: the distance-bounding grouping-proof (DBGP) protocol. The DBGP protocol is designed by combining the distance-bounding and the grouping-proof protocols, allowing for an even higher security than any of the protocols alone can provide. As seen in the security analyses, the DBGP protocols that have been designed provide the same security as the distance-bounding protocols they were built on, and almost the same security as the grouping-proof protocols (only the protection against the race condition attack was lost) they were built on.

Three DBGP protocols have been proposed. Two simpler ones that can be implemented on regular RFID tags, and a stronger one, that can be implemented on RFID tags using an ECC special-purpose processor. They show different ways of how DBGP protocols can be designed using distance-bounding and grouping-proof protocols.

A grouping-proof protocol was also designed, using the ECC technology. The security analysis of the DBGP-S protocol, built on the ECC-GP protocol, shows that the use of ECC and its techniques resulted in higher privacy for the DBGP-S protocol, when compared to the basic DBGP protocols. The ECC-GP protocol is also stronger against the grouping-proof attacks than the other grouping-proof protocols proposed in the literature, as described in Section 3.4.3.

Compared to the other, simpler grouping-proof protocols that are using the Nun function for privacy, ECC-GP is also more efficient on the reader side. After the public key is computed, it can be looked up in constant time in the database. Using the Nun function, the pseudonyms must be calculated for all tags and a linear search over all provers is needed.

5.2.2 Simulation and Test Results

The simulation results on the efficiency of the tags show that the DBGP-B1 protocol is faster than the DBGP-S protocol. This is probably affected by how the implementation is made, as the ECC computations in the DBGP-S protocol takes more time to compute than the computations for the DBGP-B1 protocol.

The small increase in the time it takes when doubling the distance-bounding rounds looks very promising. This can be interpreted as the security easily can be increased, without any major impact on the efficiency of the protocol. But, it must be noted that in the simulation framework, the communication is instantaneous. In a real application, the communication channels will be slower and the protocol may thus spend more time in the distance-bounding phase.

The false rejections simulation was rather interesting. The tests run with 1% or lower noise in the communication channel follow the expected pattern: more distance-bounding rounds makes the noise less substantial. But when testing with 10% noise in the communication channel, the false rejections increased with more rounds. This probably is due to the fact that 10% noise on the communication channel may flip the bit in *both* directions, with a probability of 10% in each direction. So, the actual probability of having the challenge or response corrupted, must be higher than 10%.

The simulation of the attacks adds up quite good with the theoretical values. Simulating the distance fraud attack against both protocols, and the mafia fraud attack against the DBGP-B1 protocol, when actually using noise on the channel gave some interesting results. With increased noise, the probability of an attack being successful decreases. This is probably because, even if the attacker has guessed the correct bit, the probability is higher that it will be corrupted. It cannot be said that the protocol is more secure in a noisy environment though, as the lower success probability is strongly related to the false rejections rate for the noise level.

It is not surprising to see that the mafia fraud attack against the DBGP-S protocol does not seem to be affected by the noise at all. An adversary launching this attack has no information about any round and must guess all responses with a probability $\frac{1}{2}$. The noise might then just as well flip a correct guessed bit as a wrong guessed bit, and does thus not change the probability at all.

5.3 Future Work

Since the first protocols were designed, new attacks against both distance-bounding and grouping-proof protocols have constantly arisen. It would therefore be relevant to anticipate that new attacks will continue to arise. If a new attack would affect any of the designed protocols, the protocol would need to be updated to be kept secure.

As the technology moves forward, the computation power on new RFID devices will increase. That may provide opportunities to use more powerful and secure methods, and the presented protocols should be updated to use these methods.

The simulation framework that has been developed provides a good foundation for simulating protocols, attacks, and noise on an RFID reader and RFID tags. It can advantageously be further developed, beneficially by someone with deeper knowledge about the RFID hardware. It would then be possible to have a more realistic environment, by adding constraints and define which methods that are allowed when implementing and simulating protocols.

6

Conclusions

A new type of protocol have been proposed. This new distance-bounding grouping-proof protocol provides the security of both the distance-bounding and the grouping-proof protocols, and can be used when the security of the either one of them by itself is not enough.

Three designed protocols of this new type have been proposed, showing that they are possible to implement and that they do provide the security of both the distance-bounding and the grouping-proof protocols they are built on.

If the results of the efficiency tests reflects how it work in reality, it is easy to motivate for more distance-bounding rounds to ensure a higher security. The results show that doubling the number of distance-bounding rounds gives a low increase in computation time, while security is highly increased, as seen in the security tests.

From the tests on the false rejections, a conclusion about tolerating noise can easily be made. Having a protocol tolerating some $X\%$ level of noise, by allowing only that $X\%$ of the distance bounding rounds to be incorrect, is not sufficient enough. It must be taken into consideration that the data may become corrupted when being sent in both directions.

It can be seen in the results from the security tests that the lower probability the DBGP-S protocol has on the mafia fraud, compared to the DBGP-B1 protocol, makes a big difference in the success probability of an adversary. The methods that are used to give this lower probability, should be employed in any protocol that want to achieve a high security.

Bibliography

[1] USPS. Intelligent Mail. URL: `https://www.usps.com/business/intelligent-mail.htm`. [Jul. 26, 2014].

[2] Daniel Hein, Johannes Wolkerstorfer, and Norbert Felber. ECC is Ready for RFID–A Proof in Silicon. In *Selected Areas in Cryptography*, pages 401–413. Springer, 2009.

[3] Yong Ki Lee, Kazuo Sakiyama, Lejla Batina, and Ingrid Verbauwhede. Elliptic-curve-based security processor for RFID. *Computers, IEEE Transactions on*, 57(11):1514–1527, 2008.

[4] Stefan Brands and David Chaum. Distance-bounding protocols. In *Advances in Cryptology-EUROCRYPT'93*, pages 344–359. Springer, 1994.

[5] Ari Juels. "Yoking-proofs" for RFID tags. In *Pervasive Computing and Communications Workshops, 2004. Proceedings of the Second IEEE Annual Conference on*, pages 138–143. IEEE, 2004.

[6] Junichiro Saito and Kouichi Sakurai. Grouping proof for RFID tags. In *Advanced Information Networking and Applications, 2005. AINA 2005. 19th International Conference on*, volume 2, pages 621–624. IEEE, 2005.

[7] Pedro Peris-Lopez, Julio C Hernandez-Castro, Juan M Estevez-Tapiador, and Arturo Ribagorda. Solving the simultaneous scanning problem anonymously: clumping proofs for RFID tags. In *Security, Privacy and Trust in Pervasive and Ubiquitous Computing, 2007. SECPerU 2007. Third International Workshop on*, pages 55–60. IEEE, 2007.

[8] Gerhard P Hancke and Markus G Kuhn. An RFID distance bounding protocol. In *Security and Privacy for Emerging Areas in Communications Networks, 2005. SecureComm 2005. First International Conference on*, pages 67–73. IEEE, 2005.

[9] Jason Reid, Juan M Gonzalez Nieto, Tee Tang, and Bouchra Senadji. Detecting relay attacks with timing-based protocols. In *Proceedings of the 2nd ACM symposium on Information, computer and communications security*, pages 204–213. ACM, 2007.

[10] Chong Hee Kim, Gildas Avoine, François Koeune, François-Xavier Standaert, and Olivier Pereira. The Swiss-knife RFID distance bounding protocol. In *Information Security and Cryptology–ICISC 2008*, pages 98–115. Springer, 2009.

[11] Marc Fischlin and Cristina Onete. Terrorism in distance bounding: modeling terrorist-fraud resistance. In *Applied Cryptography and Network Security*, pages 414–431. Springer, 2013.

[12] Ioana Boureanu, Aikaterini Mitrokotsa, and Serge Vaudenay. Secure & Lightweight Distance-Bounding. In *Second International Workshop on Lightweight Cryptography for Security & Privacy-LightSec 2013*, 2013.

[13] Yvo Desmedt. Major security problems with the 'unforgeable'(Feige)-Fiat-Shamir proofs of identity and how to overcome them. In *SecuriCom*, volume 88, pages 15–17, 1988.

[14] Cas Cremers, Kasper Bonne Rasmussen, Benedikt Schmidt, and Srdjan Capkun. Distance hijacking attacks on distance bounding protocols. In *Security and Privacy (SP), 2012 IEEE Symposium on*, pages 113–127. IEEE, 2012.

[15] Gerhard P Hancke. Distance-bounding for RFID: Effectiveness of 'terrorist fraud' in the presence of bit errors. In *RFID-Technologies and Applications (RFID-TA), 2012 IEEE International Conference on*, pages 91–96. IEEE, 2012.

[16] Leonid Bolotnyy and Gabriel Robins. Generalized "Yoking-Proofs" for a group of RFID tags. In *Mobile and Ubiquitous Systems: Networking & Services, 2006 Third Annual International Conference on*, pages 1–4. IEEE, 2006.

[17] Nai-Wei Lo and Kuo-Hui Yeh. Anonymous Coexistence Proofs for RFID Tags. *Journal of Information Science & Engineering*, 26(4), 2010.

[18] Selwyn Piramuthu. On existence proofs for multiple RFID tags. In *Pervasive Services, 2006 ACS/IEEE International Conference on*, pages 317–320. IEEE, 2006.

[19] Jung-Sik Cho, Sang-Soo Yeo, Suchul Hwang, Sang-Yong Rhee, and Sung Kwon Kim. Enhanced yoking proof protocols for RFID tags and tag

groups. In *Advanced Information Networking and Applications-Workshops, 2008. AINAW 2008. 22nd International Conference on*, pages 1591–1596. IEEE, 2008.

[20] Yuanhung Lien, Xuefei Leng, Keith Mayes, and Jung-Hui Chiu. Reading order independent grouping proof for RFID tags. In *Intelligence and Security Informatics, 2008. ISI 2008. IEEE International Conference on*, pages 128–136. IEEE, 2008.

[21] Mike Burmester, Breno De Medeiros, and Rossana Motta. Provably secure grouping-proofs for RFID tags. In *Smart Card Research and Advanced Applications*, pages 176–190. Springer, 2008.

[22] Hung-Yu Chien and Shih-Bin Liu. Tree-based RFID yoking proof. In *Networks Security, Wireless Communications and Trusted Computing, 2009. NSWCTC'09. International Conference on*, volume 1, pages 550–553. IEEE, 2009.

[23] Hsieh-Hong Huang and Cheng-Yuan Ku. A RFID grouping proof protocol for medication safety of inpatient. *Journal of Medical Systems*, 33(6):467–474, 2009.

[24] Hung-Yu Chien, Chia-Chuan Yang, Tzong-Chen Wu, and Chin-Feng Lee. Two RFID-based solutions to enhance inpatient medication safety. *Journal of Medical Systems*, 35(3):369–375, 2011.

[25] Pedro Peris-Lopez, Agustin Orfila, Julio C Hernandez-Castro, and Jan CA Van der Lubbe. Flaws on RFID grouping-proofs. Guidelines for future sound protocols. *Journal of Network and Computer Applications*, 34(3):833–845, 2011.

[26] Chih-Chung Lin, Yuan-Cheng Lai, JD Tygar, Chuan-Kai Yang, and Chi-Lung Chiang. Coexistence proof using chain of timestamps for multiple RFID tags. In *Advances in Web and Network Technologies, and Information Management*, pages 634–643. Springer, 2007.

[27] Serge Vaudenay. On Modeling Terrorist Frauds. In *Provable Security*, pages 1–20. Springer, 2013.

[28] Jens Hermans, Roel Peeters, and Cristina Onete. Efficient, secure, private distance bounding without key updates. In *Proceedings of the sixth ACM conference on Security and privacy in wireless and mobile networks*, pages 207–218. ACM, 2013.

[29] Lejla Batina, Yong Ki Lee, Stefaan Seys, Dave Singelée, and Ingrid Verbauwhede. Extending ECC-based RFID authentication protocols to privacy-preserving multi-party grouping proofs. *Personal and Ubiquitous Computing*, 16(3):323–335, 2012.

[30] Jens Hermans and Roel Peeters. Private yoking proofs: attacks, models and new provable constructions. In *Radio Frequency Identification. Security and Privacy Issues*, pages 96–108. Springer, 2013.

[31] Daniel RL Brown. Generic groups, collision resistance, and ECDSA. *Designs, Codes and Cryptography*, 35(1):119–152, 2005.

Appendices

A

Test Data

This appendix contains all the raw test data that has been used to create the plots.

A.1 Efficiency

Number of times run for each setting: 1,000.
Table A.1 shows the results for the DBGP-S protocol and Table A.2 shows the test results for the DBGP-B1 protocol.

Tags	50 DB Rounds	100 DB Rounds
3	1.65 s	1.79 s
10	5.52 s	5.76 s
25	13.52 s	14.08 s
50	26.83 s	27.36 s

Table A.1: Average time to execute the DBGP-S protocol in 1,000 executions, using different number of tags and distance-bounding rounds.

Tags	50 DB Rounds	100 DB Rounds
3	1.21 s	1.35 s
10	3.91 s	1.27 s
25	10.48 s	10.78 s
50	20.74 s	21.25 s

Table A.2: Average time to execute the DBGP-B1 protocol in 1,000 executions, using different number of tags and distance-bounding rounds.

A.2 False Rejections

Table A.3 shows the number of false rejects, out of 150,000 attempts, with different communication noise levels and different number of distance-bounding rounds.

Round	0% noise	0.01% noise	0.1% noise	1% noise	10% noise
10	0	0	13	1344	63310
20	0	0	1	437	81604
30	0	0	0	136	93275
40	0	0	0	53	101696
50	0	0	0	5	107906
60	0	0	0	2	113242
70	0	0	0	0	116994
80	0	0	0	1	120739
90	0	0	0	0	123897
100	0	0	0	0	126669

Table A.3: DF against DBGP-S protocol.

A.3 Security

A.3.1 Distance Fraud Attacks

Table A.5 and Table A.4 show the number of successful distance fraud attack attempts, out of 100,000 attempts, that were successful against the DBGP-S and the DBGP-B1 protocols, respectively.

Round	0% noise	0.01% noise	0.1% noise	1% noise	10% noise
10	24419	24572	24302	23350	15171
15	7834	7959	7943	7328	3456
20	9166	9192	9162	8313	3634
25	3077	3163	3204	2884	880
30	3746	3658	3771	3250	983
40	1649	1520	1559	1370	256
50	686	721	671	537	81
60	306	301	286	249	23
70	138	144	143	91	8
80	75	62	69	46	3
90	19	27	42	27	1
100	12	12	13	7	0

Table A.4: Successful distance fraud attacks against the DBGP-S protocol, setup to tolerate 10% noise, out of 100,000 attempts, using different number of distance-bounding rounds and levels of communication noise.

Round	0% noise	0.01% noise	0.1% noise	1% noise	10% noise
10	24151	24541	24593	23148	14835
15	7851	7953	7846	7384	3586
20	9374	9153	9147	8462	3577
25	3256	3168	3201	2780	871
30	3842	3732	3716	3206	952
40	1580	1591	1531	1329	275
50	723	675	678	547	77
60	340	344	294	255	23
70	144	147	141	113	7
80	76	49	66	42	1
90	34	41	23	16	1
100	15	15	14	12	0

Table A.5: Successful distance fraud attacks against the DBGP-B1 protocol, setup to tolerate 10% noise, out of 100,000 attempts, using different number of distance-bounding rounds and levels of communication noise.

A.3.2 Mafia Fraud Attacks

Table A.7 and Table A.6 show the number of successful mafia fraud attack attempts, out of 100,000 attempts, that were successful against the DBGP-S and the DBGP-B1 protocols, respectively.

Round	0% noise	0.01% noise	0.1% noise	1% noise	10% noise
10	1070	1145	1062	1015	1086
15	64	50	51	37	65
20	14	29	16	15	12
25	0	1	0	1	1
30	1	0	1	1	1
40	0	0	0	0	0
50	0	0	0	0	0
60	0	0	0	0	0
70	0	0	0	0	0
80	0	0	0	0	0
90	0	0	0	0	0
100	0	0	0	0	0

Table A.6: Successful mafia fraud attacks against the DBGP-S protocol, setup to tolerate 10% noise, out of 100,000 attempts, using different number of distance-bounding rounds and levels of communication noise.

Round	0% noise	0.01% noise	0.1% noise	1% noise	10% noise
10	24362	24469	24099	22636	12097
15	8035	8088	7965	7051	2570
20	8883	9068	9015	8026	2395
25	3206	3149	3141	2687	538
30	3712	3681	3718	3100	503
40	1594	1632	1565	1211	109
50	721	680	661	504	39
60	333	360	295	216	13
70	141	147	125	83	5
80	59	64	66	48	1
90	34	30	25	12	0
100	10	13	11	5	0

Table A.7: Successful mafia fraud attacks against the DBGP-B1 protocol, setup to tolerate 10% noise, out of 100,000 attempts, using different number of distance-bounding rounds and levels of communication noise.