UNIVERSITY OF TWENTE.

Faculty of Electrical Engineering, Mathematics & Computer Science

Secure biometric verification in a malicious attacker model setting

R.H.D. Scholten (Ruth)

M.Sc. Thesis
January 2020

Supervisors:
dr. A. Peter (Andreas)
dr. ir. T.A.M. Kevenaar (Tom)
prof. dr. ir. R.N.J. Veldhuis (Raymond)

Services and CyberSecurity
Faculty of Electrical Engineering,
Mathematics and Computer Science
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

Abstract

Biometric recognition is a means to measure physical characteristics of a person for authentication purposes. Physical characteristics are captured by a sensor, stored digitally in a feature vector and subsequently compared to a database template by a server. The European Union has stated in General Data Protection Regulation 2016/679 [23] that biometric data is sensitive and must therefore be protected using Biometric Template Protection (BTP) schemes. The BTP scheme constructed by Peeters et al. [56] is efficient and satisfies the requirements for a secure BTP scheme: irreversibility, unlinkability, revocability and accuracy. However, similar to most other state-of-the-art BTP schemes, it is only secure in the semi-honest model, in other words, where both the sensor and server adhere to the protocol, but may collect information along the way. In this work, we first present the semi-honest key release protocol, which follows the protocol of Peeters et al. [56], but additionally releases a key upon successful comparison. Secondly, we present the partially malicious key release protocol, which builds upon the former protocol, but uses commitment schemes and zero-knowledge proofs to provide security against a semi-honest sensor and a malicious server, i.e. a party which may deviate from the protocol. Additionally, we present the semi-honest one-round protocol: a stepping stone protocol in which the total comparison is performed by the server. The semi-honest key release protocol appears to be very practical in real-world applications, i.e. 443 comparisons per second can be done, compared to about 250 comparisons per second in the efficient semi-honest protocol of Peeters et al. [56]. The partially malicious key release protocol is less practical, i.e. 25 comparisons per second can be done. However, since runtime performance is a trade-off with accuracy, the partially malicious key release protocol can still be practical if accuracy is less important. Moreover, since the comparison operation is the bottleneck of both protocols, we expect that significant runtime efficiency can be gained here.

Acknowledgements

The past year was more or less devoted to writing my thesis. Like many theses, there were bumps on the road, but overall I enjoyed it. Now the end has come and it is time for a new journey.

First of all, I would like to thank my supervisors, Andreas Peter, Tom Kevenaar and Raymond Veldhuis, for guiding me through the master thesis project and giving me feedback on my work. Especially, I would like to thank Andreas for the helpful meetings that we had and for pointing me into new directions whenever I was stuck.

Secondly, this thesis does not only mean the end of my graduation project, but it also implies the end of my student time. During my study time I did not only learn a lot about computer science and cyber security, but I also learned a lot about myself and life in general. Therefore I would like to thank all the people I met during my time in Enschede, because I would not have become the person as I am now without you.

In particularly, I would like to thank my roommates. Thanks for being my extended family, or "gezinsvervangend tehuis" as my mother would call it in Dutch and for all the fun movie nights. As a computer science student among many guys, I really enjoyed being able to talk so much about girl stuff and let the weird side in myself come out.

I would also like to thank my friends that I met in the bachelor and master period. I made some really good friends from whom I know we will still be friends in the far future. I am also really grateful for the international friendships that I made and learn about new cultures.

During my time as a student, I was a member of RSK Enschede, a christian student association. I am really thankful for the seven years that I spent among great people whom I could share my faith with, experience so many fun things with and for everything I learned.

Last but not least, I would like to thank my family and my boyfriend Antoon for always supporting me throughout my time as a student and especially during the last year, which was not always easy. I am grateful that my parents allowed and even convinced me to be active in student life. And I am really thankful for Antoon, who took his time to proofread my thesis and who has been the most supportive person I could imagine and my best friend.

Contents

Chapter 1

Introduction

Biometric recognition is a means to measure physical characteristics (e.g. fingerprint, iris, voice, face, etc) of a person for identification or authentication purposes. Many of those characteristics are (highly) unique for each person and can thus be used as a distinctive measure to identify or authenticate a person. Physical characteristics are captured by a sensor device and stored in a digital format, called a probe. Subsequently the probe is compared to an existing biometric sample from a database by a verification server, which computes. Alternative identification or authentication methods are based on knowledge (e.g. password) or tokens (e.g. passport). These methods possess risks that biometrics do not have, such as the risk of forgetting or losing. In addition, passwords are often weak and can relatively easily be forged. Hence, biometrics is being used more and more in the field of identification and authentication [39, 56].

However, a major drawback of biometrics is the constant reuse of the same biometric in multiple authentication systems, as opposed to traditional authentication methods where multiple passwords and/or tokens are used. When a system is compromised, the biometric can never safely be used anymore in any authentication system ever again [54]. Hence, biometric templates are vulnerable to privacy problems such as identity theft [32]. The European Union has stated in General Data Protection Regulation 2016/679 [23] that biometric data is sensitive and must therefore be protected.

Protection of biometric data is handled using so-called Biometric Template Protection (BTP) schemes in which secure templates are generated from biometric samples using a variety of methods, such as encryption [32]. According to the ISO/IEC IS 24745 standard [37], a secure template should possess the following two main properties:

- *Irreversibility* or *non-invertibility*: It should be computationally hard to retrieve the original biometric sample from the secure template [31, 49].
- *Unlinkability*: It should be computationally hard to detect whether two secure templates $\mathbf{T}_1$ and $\mathbf{T}_2$ ($\mathbf{T}_1 \neq \mathbf{T}_2$), have been generated from the same biometric sample [31, 49].

Two other important properties of a secure template are:

- *Revocability*, *diversity* or *renewability*: It should be possible to revoke a secure template $\mathbf{T}_1$ when it is compromised and generate a new secure template $\mathbf{T}_2$ from the same biometric sample [38, 49, 56].
- *Accuracy* or *performance*: The accuracy or recognition performance of a biometric authentication system should not be affected by the protection of the template. In other words, the trade-off between the False Acceptance Rate (FAR) and the False Rejection Rate (FRR) should remain reasonable [38, 49, 56]. The FAR denotes the ratio of illegitimate probes falsely accepted to the system compared to the total number of verification attempts. Equivalently, the FRR denotes the ratio of legitimate probes falsely declined to the system compared to the total number of verification attempts [47].

Many BTP schemes, however, struggle with a trade-off between the mentioned properties and efficiency [56]. Nonetheless, lately many BTP schemes have been designed that possess both the mentioned properties and are efficient at the same time, such as the works of Peeters et al. [56], Sadhya et al. [58], Gomez-Barrero et al. [31, 32], Martiri et al. [48], Nazari et al. [50] and Zhang et al. [66].

1.1 Problem Statement

The BTP schemes as mentioned in the previous section are only secure against semi-honest attackers, rather than malicious attackers. In the former attacker model, a party compromised by such an attacker will not deviate from a predefined protocol, but may rather collect information, as opposed to the latter attacker model, in which an attacker performs malicious activity by not adhering to the protocol. More details on attacker models will be given in Section 2.1.3.

In this work, we will adjust the approach to secure and efficient biometric template protection as proposed by Peeters et al. [56] in such a way that it is secure in a one-sided malicious attacker setting. We chose the work of Peeters et al. [56], since the approach meets the secure template requirements as mentioned before and is based on a log-likelihood ratio classifier and elliptic curve ElGamal, which are considered state-of-the-art in terms of accuracy and efficiency, respectively.

Full malicious security will be achieved by having both parties prove their steps correctly to the other party who plays the role of a verifier. In other words, both parties will be enforced to behave in a honest manner. Unfortunately, we cannot protect biometric cryptosystems against the situation where both the sensor device and server are compromised by a (different) malicious attacker, for the obvious reason that if both parties deviate from the protocol, the proofs and verifications will be lost. For the approach by Peeters et al. [56] in the malicious model, we therefore make a case distinction:

(a) the sensor device is compromised by a malicious attacker and the server is either honest or compromised by a semi-honest attacker

(b) the sensor device is either honest or compromised by a semi-honest attacker and the server is compromised by a malicious attacker

Protecting biometric cryptosystems against both cases yields full malicious security. However, in this work, we will protect the protocol of Peeters et al. [56] only against case b, since in practice, the sensor device used in this context contains a non-tamperable capturing functionality and is thus better protected than the server. Enforcing the server to behave honest will yield a higher level of overall security in the biometric system.

Additionally, in the system of Peeters et al. [56] a simple "yes" or "no" answer is released upon comparison of a biometric probe with a database template. So called biometric-based key release systems exist that release a key when a comparison is successful. This key can for instance be used as a decryption key for sensitive data [64]. In this way there is no need to remember a decryption key and hence no risk of forgetting or weak password choices. Hence, in the current work, we will investigate and implement a way to release an key upon successful comparison of a biometric probe with a database template.

Currently, in the work of Peeters et al. [56], comparison of a biometric probe with a database template is jointly performed by the sensor device and the verification server. Although we mentioned earlier that in practice, usually, the sensor is better protected than the server, there are also cases in which there is no clear supervision on the secure setup of sensor devices. In these cases the only way to provide a better level of security is to protect the server against malicious attackers and have it do as many of the computations in the protocol as possible.

1.2 Research Questions

From the problem statement, we have deducted the following research questions. First of all, we will adapt the semi-honest protocol by Peeters et al. [56] such that it releases a key upon successful comparison, hence the following research question:

RQ1 How can we adapt the approach by Peeters et al. [56] in such a way that a key is released upon successful recognition instead of a simple "yes" or "no"?

From here on, we refer to the protocol as constructed in RQ1 as the *semi-honest key release protocol*.

Secondly, we make the semi-honest key release protocol secure against a malicious verification server. Hence, we will answer the following research question, which is the main focus of this work:

RQ2 How can we adapt the semi-honest key release protocol in such a way that it is secure in the setting where the sensor device is (semi-)honest and the server is malicious?

From here on, we refer to the protocol as constructed in RQ2 as the *partially malicious key release protocol*.

Subsequently, we will dive deeper into adjusting the semi-honest key release protocol such that the comparison is not a joint effort from the sensor and server anymore, but a black box from the sensor's point of view and completely performed by the server. Accordingly, we will answer the following research question:

RQ3 How can we transform the semi-honest key release protocol in such a way that it is a one-round protocol in which comparison is done on the server side?

From here on, we will refer to the protocol as constructed in RQ3 as the *semi-honest one-round protocol*.

Finally, we will assess the performance of the partially malicious key release protocol in terms of the number of operations and runtime and compare the performance to the work of Peeters et al. [56] and the semi-honest key release protocol as constructed in this research. To measure the runtime, we will create proof of concepts for both the semi-honest key release protocol and the partially malicious key release protocol. Hence, we will answer the following research question:

RQ4 How efficient is the partially malicious key release protocol and how does the partially malicious key release protocol compare to related semi-honest and malicious BTP schemes in terms of efficiency?

1.3 Contribution

To give a summary of the contributions in this work, we present the following three protocols:

- The semi-honest key release protocol, which uses the biometric recognition protocol by Peeters et al. [56] as a basis and binds a key to a biometric template which is released upon successful comparison. The key is independent from the biometric template and can be revoked. The semi-honest key release protocol is based on the NIST K-233 elliptic curve and uses ElGamal for efficiency purposes. The semi-honest key release protocol can process 443 biometric template comparisons per second, compared to 250 comparisons per second in the work of Peeters et al. [56], which is considered practical in realistic settings.

- The partially malicious key release protocol, which builds upon the semi-honest key release protocol and enforces a maliciously compromised server to behave in an semi-honest fashion, using commitment schemes and zero-knowledge proofs. Moreover, we haven chosen the protection mechanisms in such a way that extending the work to the full malicious model is rather straightforward. The partially malicious key release protocol is less practical than the semi-honest key release protocol, i.e. 25 comparisons per second can be done and therefore only practical if the accuracy is less important.

- The semi-honest one-round protocol, in which the sensor device sends its identity claim and probe vector to the server, and the server performs a comparison and sends the result back to the sensor device. We only give a theoretic overview of the protocol, therefore we consider the semi-honest one-round protocol as a stepping stone for further research in the area of one-round protocols.

1.4 Outline of Thesis

This paper is structured as follows. First, in Chapter 2 we present the background information needed to understand the rest of this work. Secondly, in Chapter 3, we discuss some related work and argue why this paper has scientific relevance. Thirdly, in Chapter 4, we describe the semi-honest protocol by Peeters et al. [56], which forms the basic ground of the following chapters. In Chapter 5, we present the semi-honest key release protocol and in Chapter 6 we present the partially malicious key release protocol. Then in Chapter 7, we give a theoretic overview of the semi-honest one-round protocol. Subsequently, in Chapter 8 we provide an efficiency analysis of the partially malicious key release protocol. Finally, in Chapter 9 we give the conclusion and limitations of this work, and discuss elements for future work.

Chapter 2

Background

In this chapter, we will elaborate on the preliminary knowledge needed to understand the subsequent chapters. First, in Section 2.1 we present the most important definitions used in this work. Then, in Section 2.2, we describe some important cryptographic primitives. Finally, in Section 2.3, we explain the cryptographic protocols which form the basis for the upcoming protocols in this work.

2.1 Definitions

In order to understand the notation used in this work, we first briefly present some cryptographic notations in Section 2.1.1. Then, in Sections 2.1.2 and 2.1.3, we give formal and informal definitions of security and the two most essential attacker models used in secure multi-party cryptography.

2.1.1 Cryptographic Notations

In this work, we use the following cryptographic notations:

Notation	Description
pk_{shared}	shared public key between server and sensor obtained by threshold encryption
pk_{ss}	public key of sensor device
pk_{sv}	public key of server
sk_{ss}	secret key of sensor device
sk_{sv}	secret key of server
$\mathsf{E}(m)$	encryption function on message m under public key pk_{shared}
$\mathsf{E}_{pk}(m)$	encryption function on message m under some public key pk
$\mathsf{D}_1(c)$	decryption of ciphertext c under sk_{sv}
$\mathsf{D}_2(c)$	decryption of ciphertext c under sk_{ss}
$\mathsf{AES}_K(c)$	AES encryption on ciphertext c using key K
$\mathsf{AES\text{-}D}_K(c)$	AES decryption of ciphertext c under key K
$[m]_{\mathsf{pk}}$	encryption of message m under some public key pk
$[m]$	encryption of message m under public key pk_{ss}
$[[m]_{\mathsf{pk}_1}]_{\mathsf{pk}_2}$	double-encryption of message m, first under public key pk_1 and then under public key pk_2
$[[m]]$	double-encryption of message m under public key pk_{shared}
$\mathsf{Com}(m, r)$	commitment over message m using randomness r
$\mathsf{Com}(m)$	commitment over message m using some uniformly distributed randomness
$\mathsf{Decom}(m)$	decommitment or reveal of message m
$\mathsf{PK}(m)$	zero-knowledge proof-of-knowledge on input m
$\mathsf{BCT}()$	execution of Basic Coin-Tossing protocol
$\mathsf{AC}()$	execution of Authenticated Computation protocol
$\mathsf{ACT}()$	execution of Augmented Coin-Tossing protocol
$\stackrel{c}{\equiv}$	computational indistinguishability
$\mathcal{M}$	message space in the corresponding context

2.1.2 Security

Security in a multi-party computation setting can be defined by using the real/ideal paradigm. Informally, in the ideal model, for a two-party computation setting, two parties send their inputs (x, y) to a trusted party who computes result $f(x, y) = (f_1(x, y), f_2(x, y))$ and answers party 1 and 2 with $f_1(x, y)$ and $f_2(x, y)$, respectively [28]. In the real model, parties jointly run a protocol without a trusted party. Now the notion of security can be informally defined as follows [46]:

Definition 1. Let A be an attacker in the real model and S an attacker in the ideal model.

A protocol π computes a function f in a secure way if $\forall A \rightarrow \exists S$, such that any result of an execution of π with A in the real model is computationally indistinguishable from an execution of π with S in the ideal model.

2.1.3 Attacker Models

In this work, we distinguish between two types of attackers or adversaries: *semi-honest attackers* and *malicious attackers* [30]. We assume that if a party acts according to a certain adversarial model, it is compromised by an attacker of such kind.

2.1.3.1 Semi-Honest Attacker

A semi-honest attacker, also called honest-but-curious or passive attacker is an attacker who compromises a system and collects data as the system runs its protocol. The semi-honest attacker is passive in the sense that it does not intervene the system's protocol run, i.e. the system continues to follow its protocol correctly. We assume that a semi-honest attacker is static, i.e. its attacker model does not change throughout the course of the protocol run [28].

Formally, we can define security in the semi-honest model for a general function as follows [28, 45]:

Definition 2. A function f securely computes protocol π in the semi-honest model if there exist probabilistic polynomial-time algorithms, S_1 and S_2, such that:

$$\{(S_1(1^n, x, f_1(x, y)), f(x, y))\}_{x,y,n} \stackrel{c}{\equiv} \{(\mathsf{VIEW}_1^\pi(x, y, n), \mathsf{OUTPUT}^\pi(x, y, n))\}_{x,y,n} \tag{2.1}$$

$$\{(S_2(1^n, y, f_2(x, y)), f(x, y))\}_{x,y,n} \stackrel{c}{\equiv} \{(\mathsf{VIEW}_2^\pi(x, y, n), \mathsf{OUTPUT}^\pi(x, y, n))\}_{x,y,n} \tag{2.2}$$

for secret inputs $x, y \in \{0, 1\}^*$, $|x| = |y|$, security parameter $n \in \mathbb{N}$ and function outputs $f_1(x, y)$ and $f_2(x, y)$. $\mathsf{VIEW}_i^\pi(x, y, n)$ and $\mathsf{OUTPUT}_i^\pi(x, y, n)$ denote the view and output of party $i \in \{1, 2\}$ on protocol π, respectively. Relation operator $\stackrel{c}{\equiv}$ indicates computational indistinguishability [45].

In other words, there exists an algorithm or simulator S_i for $i \in \{1, 2\}$ in the ideal model execution of π that is, given the input and the output of S_i and the general function output, computationally indistinguishable from the view of Party i and the protocol output in the real model execution of π [45].

In case functionality f is deterministic, we can adopt a more apparent notion of security in the semi-honest model [28, 45]:

Definition 3. A deterministic function f securely computes protocol π in the semi-honest model if there exist probabilistic polynomial-time algorithms, S_1 and S_2, such that:

$$\{S_1(1^n, x, f_1(x, y))\}_{x,y\in\{0,1\}^*;n\in\mathbb{N}} \stackrel{c}{\equiv} \{\mathsf{VIEW}_1^\pi(x, y, n)\}_{x,y\in\{0,1\}^*;n\in\mathbb{N}} \tag{2.3}$$

$$\{S_2(1^n, y, f_2(x, y))\}_{x,y\in\{0,1\}^*;n\in\mathbb{N}} \stackrel{c}{\equiv} \{\mathsf{VIEW}_2^\pi(x, y, n)\}_{x,y\in\{0,1\}^*;n\in\mathbb{N}} \tag{2.4}$$

In other words, there exists an algorithm or simulator S_i for $i \in \{1, 2\}$ in the ideal model execution of π that, given the input and the output of S_i, is computationally indistinguishable from the view of Party i in the real model execution of π [45].

However, Definition 3 cannot be used in the context of probabilistic functions, since for a function that has randomized output it does not always apply that $\mathsf{OUTPUT}^\pi(x, y, n) = f(x, y)$ for each set of input values (x, y) [28]. Therefore, we need to include the simulator's output and the output of the protocol in Definition 2.

For a deterministic functionality f, Definition 3 implies Definition 2 if also a correctness proof is given.

Formally, we can define correctness of a function f as follows [45]:

$$\Pr[\mathsf{OUTPUT}^{\pi}(x, y, n) \neq f(x, y)] \leq \mu(n) \tag{2.5}$$

for a negligible function $\mu(n)$.

Now, if a protocol computing a deterministic function conforms to the correctness condition, it inevitably means that $\mathsf{OUTPUT}^{\pi}(x, y, n) \stackrel{c}{\equiv} f(x, y)$ and thus [45]:

$$\{(\mathsf{VIEW}_1^{\pi}(x, y, n), f(x, y))\}_{x,y\in\{0,1\}^*;n\in\mathbb{N}} \stackrel{c}{\equiv} \{(\mathsf{VIEW}_1^{\pi}(x, y, n), \mathsf{OUTPUT}^{\pi}(x, y, n))\}_{x,y\in\{0,1\}^*;n\in\mathbb{N}} \tag{2.6}$$

Moreover, a simulator can compute functionality $f(x, y)$, since he is given x and y. Therefore Equation 2.3 implies [45]:

$$\{(S_1(1^n, x, f_1(x, y)), f(x, y))\}_{x,y\in\{0,1\}^*;n\in\mathbb{N}} \stackrel{c}{\equiv} \{(\mathsf{VIEW}_1^{\pi}(x, y, n), f(x, y))\}_{x,y\in\{0,1\}^*;n\in\mathbb{N}} \tag{2.7}$$

Equations 2.6 and 2.7 together yield:

$$\{(S_1(1^n, x, f_1(x, y)), f(x, y))\}_{x,y\in\{0,1\}^*;n\in\mathbb{N}} \stackrel{c}{\equiv} \{(\mathsf{VIEW}_1^{\pi}(x, y, n), \mathsf{OUTPUT}^{\pi}(x, y, n))\}_{x,y\in\{0,1\}^*;n\in\mathbb{N}} \tag{2.8}$$

and thus Definition 3 implies Definition 2 for a deterministic function. For more information, we refer to the tutorial on the simulation proof technique by Lindell [45].

2.1.3.2 Malicious Attacker

A malicious attacker, also called an active attacker, is an attacker who compromises a system and takes control of its protocol run for malicious purposes. The malicious attacker may corrupt data or abort the protocol for instance, i.e. a compromised system may deviate from its instructed protocol. In this work, we make the following assumptions about malicious attackers [28, 30]:

- Attacker modes are dynamic, i.e. any party can become a malicious attacker during a protocol run.
- Since our approach is a two-party protocol, we assume no collusion will take place. In case the server and sensor are both compromised by a malicious attacker, there is not much we can do. Therefore we assume either of the two parties is allowed to be malicious and the other should remain honest or semi-honest throughout the protocol run.
- If a (semi-)honest party detects malicious behavior from another party, it will abort the protocol.

A malicious attacker can engage in the following (malicious) activity [28]:

- A malicious party may participate in the protocol with input $x' \in \{0,1\}^{|x|}$, where x is the original input given to the party and $x' \neq x$ applies.
- A malicious party might abort the protocol at any time, even when the protocol has not finished. This means that a malicious party can abort when he receives the output of the protocol, even before the other (honest) party receives the output. Thus, two parties obtaining the result of a computation protocol synchronously (perfect fairness) is not possible in a two-party computation protocol. A malicious party can also abort even before the protocol starts.

We can define malicious attackers in the ideal model as follows [28]:

Definition 4. Let functionality f be defined as follows: $f : 0,1^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$, where $f_1(x, y)$ and $f_2(x, y)$ represent outputs of f for the first and the second party, respectively. Furthermore, let $\overline{B} = (B_1, B_2)$ be a pair of probabilistic polynomial-time algorithms representing strategies in the ideal model. $\overline{B}$ is admissible in the ideal malicious model if $B_i(u, z, r) = u$ and $B_i(u, z, r, v) = v$ for $\forall u, z, r, v$ where there is at least one honest party $i \in \{1, 2\}$ and $|B_i(u, z, r)| = |u| \mid \forall i$ applies. The joint execution of f under $\overline{B}$ in the ideal model with input pair (x, y) and auxiliary input z can be denoted as $IDEAL_{f,\overline{B}(z)(x,y)} = \Upsilon(x, y, z, r)$ where r is a uniformly chosen random-tape for the adversary. Now $\Upsilon(x, y, z, r)$ can be defined as follows:

If Party 1 is honest, then:

$$\Upsilon(x, y, z, r) = (f_1(x, y'), B_2(y, z, r, f_2(x, y'))), \text{ where } y' = B_2(y, z, r) \tag{2.9}$$

If Party 2 is honest, then:

$$\Upsilon(x, y, z, r) = \begin{cases} (B_1(x, z, r, f_1(x', y), \perp), \perp) & \text{if } B_1(x, z, r, f_1(x', y)) = \perp \\ (B_1(x, z, r, f_1(x', y)), f_2(x', y)) & \text{otherwise} \end{cases} \tag{2.10}$$

where $x' = B_1(x, z, r)$.

Equation 2.10 refers to the case where Party 1 enters the protocol by calling the trusted party with input $B_1(x, z, r)$. Since Party 1 may be malicious, its input might be substituted. In case $B_1(x, z, r, f_1(x', y)) = \perp$, Party 1 aborts the protocol after he received his output $f_1(B_1(x, z, r), y)$, before Party 2 receives his output from the trusted party. Otherwise, Party 1 does not abort the protocol and thus Party 2 receives his output $f_2(B_1(x, z, r), y)$ from the trusted party. In the case of Equation 2.9, Party 2 may be malicious, but Party 1 obtains his output first from the trusted party, and thus there is no way for Party 2 to prevent the trusted party from answering Party 1.

We can define malicious attackers in the real model as follows [28]:

Definition 5. Let f be defined as in Definition 4 and let π be two-party protocol to compute f. Now, let $\overline{A} = (A_1, A_2)$ be a pair of probabilistic polynomial-time algorithms representing strategies in the real model. $\overline{A}$ is admissible in the real malicious model, if at least one A_i correspond with the strategy as determined by π, which means that A_i ignores auxiliary input z. The joint execution of π under $\overline{A}$ in the real model with input pair (x, y) and auxiliary input z can be denoted as $REAL_{\pi,\overline{A}(z)(x,y)}$ which defines the output pair which emerges from the interaction between $A_1(x, z)$ and $A_2(y, z)$.

Now we can define security in the malicious model as follows [28]:

Definition 6. A deterministic function f securely computes protocol π in the malicious model if for each pair of probabilistic polynomial-time algorithms $\overline{A} = (A_1, A_2)$ that are admissible for the real model, there exists a pair of probabilistic polynomial-time algorithms $\overline{B} = (B_1, B_2)$ that are admissible for the ideal model, such that:

$$\{\text{IDEAL}_{f,\overline{B}(z)}(x, y)\}_{x,y,z} \stackrel{c}{\equiv} \{\text{REAL}_{\pi,\overline{A}(z)}(x, y)\}_{x,y,z} \tag{2.11}$$

where $x, y, z \in \{0, 1\}^*$ such that $|x| = |y|$ and $|z| = poly(|x|)$.

In other words, the emulation of a secure protocol π in the real model is computationally indistinguishable from the computation of functionality f in the ideal model.

2.2 Cryptographic Primitives

In this section, we present the type of encryption we use in this work: the ElGamal encryption system (Section 2.2.1), homomorphic encryption (Section 2.2.2) and elliptic curve cryptography (Section 2.2.3).

2.2.1 ElGamal Encryption System

The ElGamal encryption system is based on the discrete logarithm problem, which means that given a cyclic group $\mathcal{G}$ with generator g, it is computationally hard to retrieve x for g^x.

Given cyclic group $\mathcal{G} \subseteq \mathbb{Z}_p^*$ with generator $\langle g \rangle = \mathcal{G}$, order $q = |\mathcal{G}|$, such that $p, q \in \mathbb{P}$, $q|p - 1$, secret key $a \in_R \mathbb{Z}_q$ and public key $h = g^a \mod p$, the encryption function on a message $m \in \mathcal{G}$ can be defined as follows [56, 62]):

$$c = (c_1, c_2) = (g^r, m \cdot h^r) \tag{2.12}$$

where $r \in_R \mathbb{Z}_q$.

For each encryption of message m, the resulting ciphertext is different, due to randomness r, hence why ElGamal is probabilistic. Moreover, ElGamal is secure under a chosen plaintext attack, meaning that, given the ciphertext, it is computationally hard for an attacker to recover the plaintext, assuming the Diffie-Hellman problem is hard [62].

A ciphertext $c = (c_1, c_2)$ can be decrypted as follows:

$$\frac{c_2}{c_1^a} = \frac{m \cdot h^r}{(g^r)^a} = \frac{m \cdot (g^a)^r}{(g^r)^a} = \frac{m \cdot g^{ar}}{g^{ar}} = m \tag{2.13}$$

Similar to the work of Peeters et al. [56], we chose to encode a message m as exponent of generator g to support additions of similarity scores. See Section 2.2.2 for further explanation.

2.2.2 Homomorphic Encryption

Homomorphic encryption schemes are encryption schemes that allow computations in the encrypted domain such as addition and multiplication to be performed on ciphertexts.

Homomorphic encryption schemes can support multiplicative, additive or both operations in the encrypted domain. In case a homomorphic encryption scheme supports multiplicative operations, the following operation can be performed. Assume that we have two messages m_1 and m_2, then:

$$[m_1] \cdot [m_2] = [m_1 \cdot m_2] \tag{2.14}$$

ElGamal is multiplicatively homomorphic as can be shown as follows. Assume that we have two ElGamal encryptions $c = (c_1, c_2)$ and $c' = (c'_1, c'_2)$ generated from the same group under the same public key. Then the following applies:

$$\begin{aligned} c \cdot c' &= (g^r, m \cdot h^r)(g^{r'}, m' \cdot h^{r'}) \\ &= (g^r g^{r'}, (m \cdot h^r)(m' \cdot h^{r'})) \\ &= (g^{r+r'}, (m \cdot m')h^{r+r'}) \\ &= (c_1 \cdot c'_1, c_2 \cdot c'_2) \end{aligned} \tag{2.15}$$

If a homomorphic encryption scheme supports additive operations, then the following operation can be performed:

$$[m_1] * [m_2] = [m_1 + m_2] \tag{2.16}$$

where $*$ denotes an arithmetic operation, such as multiplication or addition.

ElGamal is also additively homomorphic if we encode a message m as g^m, as discussed earlier in Section 2.2.1. Recall that we are not able to retrieve m if we only have g^m, due to the discrete logarithm problem [56]. Now we can add two encrypted messages as follows, which we call homomorphic addition:

$$[g^{m_1}] \cdot [g^{m_2}] = [g^{m_1+m_2}] \tag{2.17}$$

An example for ElGamal encryptions c and c':

$$\begin{aligned} c \cdot c' &= (g^r, g^m h^r)(g^{r'}, g^{m'} h^{r'}) \\ &= (g^r g^{r'}, (g^m h^r)(g^{m'} h^{r'})) \\ &= (g^{r+r'}, g^{m+m'} h^{r+r'}) \end{aligned} \tag{2.18}$$

In case we need to multiply a message m, which has been encrypted, by a specific factor k in the context of additively homomorphic ElGamal, we perform a scalar multiplication as follows:

$$[g^m]^k = [g^{m \cdot k}] \tag{2.19}$$

An example for ElGamal encryption c and factor k:

$$c^k = (g^r, g^m \cdot h^r)^k = (g^{r \cdot k}, g^{m \cdot k} h^{r \cdot k}) \tag{2.20}$$

2.2.3 Elliptic Curve Cryptography

Many public key cryptosystems, such as ElGamal, are based on elliptic curves over finite fields, hence we call this approach elliptic curve cryptography (ECC). In this section, we will cover the basics of ECC, i.e. the operations on an elliptic curve, since we will express the computational complexity of the protocols in this work in the number of elliptic curve multiplications.

ECC provides the same security as cryptosystems which are directly based on cyclic groups, however its finite field can be chosen smaller, since it is assumed that the discrete logarithm problem is harder to solve for elliptic curves than for prime integers. As a result, the performance in terms of implementation runtime is better for ECC [21, 42].

Elliptic curves can be defined as follows. In case the elliptic curve is based on Galois field $\mathbb{F}_p$ where p is a sufficiently large prime, we can define its solution set in the form of a simplified Weierstrass equation as follows [21]:

$$E(\mathbb{F}_p) : y^2 = x^3 + ax + b \pmod{p} \tag{2.21}$$

where $a, b \in \mathbb{F}_p$ and $4a^3 + 27b^3 \neq 0$.

In case the elliptic curve is based on a binary field $\mathbb{F}_{2^n}$, we can define the corresponding simplified Weierstrass equation as follows [21]:

$$E(\mathbb{F}_{2^n}) : y^2 + xy = x^3 + ax^2 + b \tag{2.22}$$

where $a, b \in \mathbb{F}_{2^n}$ for $n \in \mathbb{P}$ and $b \neq 0$.

A special variant of an elliptic curve over a binary field is the Koblitz curve, where $a = \{0, 1\}$ and $b = 1$. The parameters of Koblitz curves have been chosen such that operations on the elliptic curve can be done more efficiently than standard curves over a prime or other binary fields [41]. In this work, we will make use of a Koblitz curve.

E forms an abelian group for specific values of a and b and when the point at infinity $\mathcal{O}$ is added to it. Since E is now commutative, associative and has an identity element and an inverse for each element by definition, E can be used as a discrete logarithm group for public key cryptosystems [59].

For points $P = (x_1, y_1), Q = (x_2, y_2) \in \mathbb{F}_{2^n}$ where $P, Q \neq \mathcal{O}$, we can compute *point addition* $R = (x_3, y_3) = P + Q$ as follows:

$$\begin{aligned} x_3 &= \lambda^2 + \lambda + x_1 + x_2 + a \\ y_3 &= \lambda(x_1 + x_3) + x_3 + y_1 \end{aligned} \tag{2.23}$$

where in the case of $P \neq \pm Q$, we define λ as [21]:

$$\lambda = \frac{y_2 + y_1}{x_2 + x_1} \tag{2.24}$$

We refer to the case where $P = Q$ as *point doubling*. In this case λ is computed as [21]:

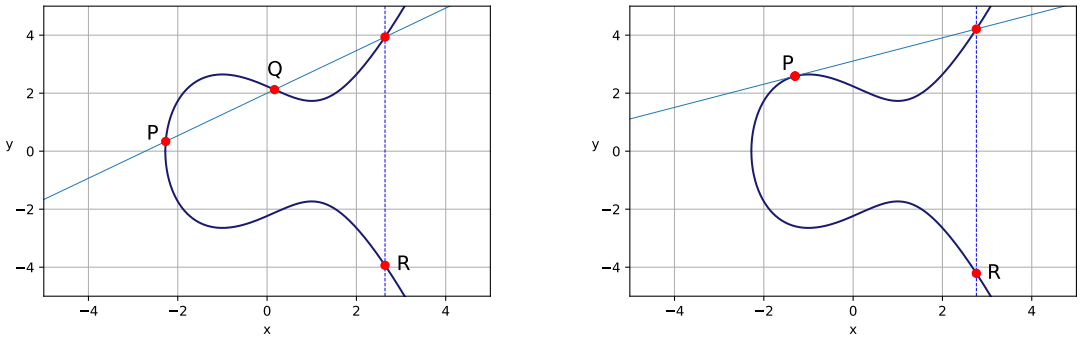
$$\lambda = x_1 + \frac{y_1}{x_1} \tag{2.25}$$

Also, a point $P = (x_1, y_1)$ can be negated as follows [43]:

$$-P = (x_1, -y_1) \tag{2.26}$$

Scalar point multiplication in the form of $R = mP$, where $m \in \mathbb{N}$, can be done using a repeated combination of point addition and point doubling, called the double-and-add method, which is similar to the square-and-multiply algorithm. For example, $R = 11P$ can be computed as $R = 2(2(2P) + P) + P$ [59].

To give the reader a feeling of how elliptic curves and the operations on them look like, we have plotted a simple example of point addition and doubling in Figure 2.1 below.



(a) Point addition

(b) Point doubling

Figure 2.1: Operations on curve $y^2 = x^3 - 3x + 5$

In Figure 2.1a, point $R = P + Q$ is obtained by drawing a line through P and Q, extracting the third point of intersection with the elliptic curve and reflecting this point on the x-axis. In Figure 2.1b, point $R = 2P$ is obtained by drawing the tangent line of P on the curve, extracting the second point of intersection with the curve and reflecting this point on the x-axis.

2.2.3.1 EC-ElGamal

In this work, we use EC-ElGamal, i.e. elliptic curve based ElGamal. We can describe the additively homomorphic EC-ElGamal cryptosystem as follows [43]:

Given a Galois field $\mathbb{F}_{2^m}$ with generator G, order q, curve E, secret key $a \in_R \mathbb{Z}_q$ and public key $H = aG$, we can define the encryption function for EC-ElGamal on a message $m \in \mathbb{Z}_q$ as:

$$C = (P, Q) \leftarrow (rG, mG + rH) \tag{2.27}$$

where $r \in_R \mathbb{Z}_q$.

The decryption function on a ciphertext $C = (P, Q)$ can now be defined as:

$$mG \leftarrow Q - aP \tag{2.28}$$

Observe that from the decryption function, we cannot retrieve message m due to the discrete logarithm problem, similar to the additively homomorphic version of plain ElGamal, where m is encoded as g^m.

The additions and multiplications in the encryption and decryption function can be done using point additions, negations and scalar point multiplications as described in the previous section.

Despite the fact that in this work we use additively homomorphic EC-ElGamal, we continue to use the additively homomorphic ElGamal notation as described in Section 2.2.1 for simplicity purposes. However, the use of terms as multiplications and additions in the context of EC-ElGamal and ElGamal might be confusing, since they have different meanings in both contexts. We therefore use the terms *scalar point multiplication* and *point addition* when dealing with EC-ElGamal and *scalar multiplication* and *homomorphic addition* when dealing with ElGamal. Also note that in the context of additively homomorphic ElGamal we use the terms scalar multiplication and ciphertext multiplication, and homomorphic addition and ciphertext multiplication interchangeably.

2.3 Basic Cryptographic Protocols

In this section, we explain the two most fundamental cryptographic protocols used in this work: commitment schemes (Section 2.3.1) and zero-knowledge proofs (Section 2.3.2).

2.3.1 Commitment Schemes

Commitment schemes are the building blocks of many cryptographic protocols. In a basic commitment scheme, one party called Alice, sends a message to a second party, Bob. However, since Alice and Bob do not trust each other, Alice should not be able to alter her message during the protocol stage, hence Alice commits to her message. Bob should not learn the message, until Alice decommits, in other

words, reveals her message. Then, Bob can verify whether the message sent by Alice was originally constructed by her [19, 29].

More formally, we can define a commitment scheme as follows [7]:

Definition 7. First of all, assume that a group $\mathcal{G}$ of prime order for which the discrete logarithm problem is hard has been established. Now a commitment scheme should contain the following three algorithms:

- $\mathsf{pk} \leftarrow \mathsf{KeyGen}(1^\ell)$: Key generation algorithm KeyGen with input 1^ℓ, where ℓ is a security parameter, produces a public key pk used for the commitment.
- $(c, d) \leftarrow \mathsf{Com}(\mathsf{pk}, m, r)$: Using public key pk and a randomness r, commitment algorithm Com generates a commitment upon message $m \in \mathcal{M}$, where $\mathcal{M}$ is the message space in the current context. Additionally, it produces decommitment message $d = (m, r)$. For simplicity purposes, we will disregard input value pk from here on.
- $\{\mathsf{accept}, \mathsf{reject}\} \leftarrow \mathsf{Verify}(\mathsf{pk}, c, d, m)$: Verification algorithm Verify has as input public key pk, a commitment c, decommitment message d and message m and returns accept or reject.

Moreover, a commitment scheme should have the binding and hiding properties. Binding means that the party who committed to a certain value cannot change this value anymore. Hence, when the value is revealed, we can be certain that this is the same value the party originally chose [19]. More formally, we can write the binding property as follows [62]:

Definition 8. A commitment scheme is computationally (resp. perfectly) binding if a computationally bounded (resp. unbounded) adversary creates a commitment c for message $m \in \mathcal{M}_1$ and randomness $r \in \mathcal{M}_2$ and is unable to find an $m' \in \mathcal{M}_1$ $(m' \neq m)$ and a randomness $r' \in \mathcal{M}_2$ for which the following holds:

$$\mathsf{Com}(m, r) = \mathsf{Com}(m', r') \tag{2.29}$$

Schematically, we can represent the binding game as follows [62]:

Adversary		**Challenger**
Input: $m \in \mathcal{M}_1, r \in \mathcal{M}_2, (c, d) \leftarrow \mathsf{Com}(m, r)$		
	$\xrightarrow{m, r, c}$	
$m' \in \mathcal{M}_1 \quad (m' \neq m)$ $r' \in \mathcal{M}_2$ $c' = \mathsf{Com}(m', r')$		
	$\xrightarrow{m', r'}$	
		Output: $\begin{cases} \text{A wins,} & \text{if } \mathsf{Com}(m, r) = \mathsf{Com}(m', r') \\ \text{A loses,} & \text{otherwise} \end{cases}$

Figure 2.2: Binding game of a commitment scheme

Hiding means that Bob is unable to know the value which has been committed by Alice [19], or in other words, the value committed by Alice is indistinguishable from another committed value [62]. More formally, we can write the hiding property as follows [62]:

Definition 9. A commitment scheme is computationally (resp. perfectly) hiding if an adversary is able to win the following game:

- The adversary computes messages $m_0, m_1 \in \mathcal{M}_1$ of the same length.
- The challenger produces randomness $r \in_R \mathcal{M}_2$ and bit $d \in_R \{0, 1\}$.
- The challenger computes commitment $c = \mathsf{Com}(m_d, r)$ and sends c to the adversary.
- The adversary tries to guess d.

Schematically, we can represent the hiding game as follows [62]:

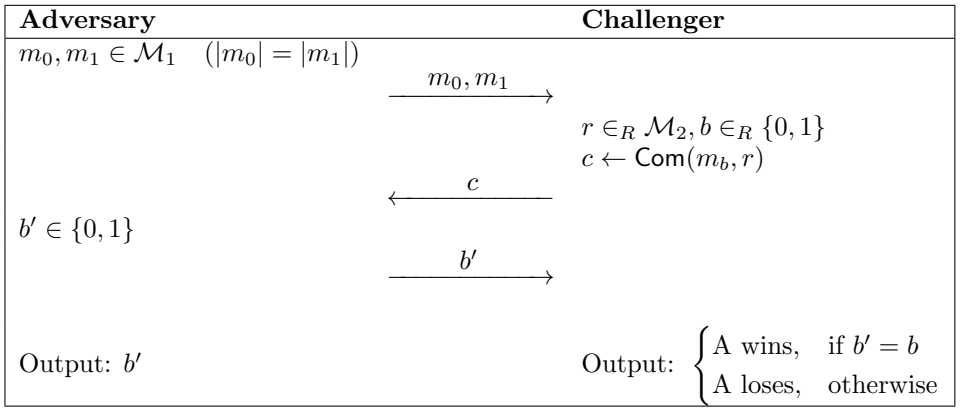


Figure 2.3: Hiding game of a commitment scheme

Essentially, perfectly binding or hiding means that no computational resources exist to break the commitment scheme. In the same way, computational hiding or binding means computational resources exist to break the commitment scheme. However, we consider these resources to be extremely expensive in terms of efficiency, therefore, the probability to break the commitment scheme in case of computational hiding or binding is negligible [19].

Pedersen Commitment Scheme In this work, we use a commitment scheme as constructed by Pedersen [55].

Assume that the Pedersen commitment scheme works over group $\mathcal{G} \subseteq \mathbb{Z}_p^*$ with order q and generator g, where p and q are prime and $q|p-1$ applies. The commitment algorithms can now be instantiated as follows:

$$\mathsf{KeyGen}(1^\ell) = h \tag{2.30}$$

where $\langle h \rangle = \mathcal{G}$.

$$\mathsf{Com}(m, r) = g^m \cdot h^r \tag{2.31}$$

where $m \in \mathbb{Z}_q$ and $r \in_R \mathbb{Z}_q$.

$$\mathsf{Verify}(\mathsf{pk}, c, d, m) : \mathsf{Com}(d) \stackrel{?}{=} c \tag{2.32}$$

The Pedersen commitment scheme is computationally binding due to the hardness of the discrete logarithm problem and perfectly hiding, because the commitment is uniformly distributed over $\mathcal{G}$, i.e. different sets of (m, r) exist that satisfy the commitment [22].

2.3.2 Zero-Knowledge Proof-of-Knowledge

A zero-knowledge proof-of-knowledge in cryptography is used where a party (prover) needs to prove to another party (verifier) that it knows a certain value, called a witness, without revealing anything but the fact that the prover possesses this knowledge.

2.3.2.1 Σ-Protocol

A zero-knowledge protocol can be simulated by a so-called Σ-protocol [62]. A Σ-protocol is a two-party protocol in which a prover and a verifier perform a 3-move protocol to perform a zero-knowledge proof-of-knowledge. Σ-protocols should satisfy completeness, special soundness and special honest-verifier zero-knowledge. Completeness implies that if a prover P knows witness w, the probability that the verifier will reject the proof is negligible. Special soundness implies that if there are two protocol runs ρ and ρ' for which $t = t'$, but $c \neq c'$ applies, then the prover's witness can be retrieved. Special honest-verifier zero-knowledge essentially means that as long as the verifier is (semi-)honest, it does not learn anything about the witness, other than that the prover knows this witness. [19, 62].

Formally, we can define a Σ-protocol as [7, 36]:

Definition 10. Let π be a two-party protocol between a prover and a verifier, where the verifier is probabilistic polynomial time bounded, and let $\mathcal{R}$ be a binary relation where $\mathcal{R} \subset \{0,1\}^* \times \{0,1\}^*$. Then protocol π is a Σ-protocol for $\mathcal{R}$ with challenge set $\mathcal{C}$, common input y and private witness w for the prover, if and only if it holds the following properties:

- π is a *3-move protocol* as follows:
  - The prover computes a commitment t and sends t to the verifier.
  - The verifier chooses a challenge $c \in \mathcal{C}$ with challenge length m and sends c to the prover. m is denoted as the soundness parameter and we define the knowledge error of π as 2^{-m}.
  - The prover computes a response s and sends s to the verifier.
  - The verifier outputs accept or reject based on the combination of the commitment, challenge and response.
- *Completeness*: If $y, w \in \mathcal{R}$, then the verifier accepts with probability $1 - \mu(n)$, where $\mu(n)$ is a negligible function on input $n \in \mathbb{N}$.
- *Special soundness*: There exists a probabilistic polynomial time knowledge extractor E with inputs protocol runs $\rho = (t, c, s)$ and $\rho' = (t', c', s')$ for which $t = t'$ and $c \neq c'$ applies, and an output w' such that $(y, w') \in \mathcal{R}$.
- *Special honest-verifier zero-knowledge*: There exists a probabilistic polynomial time simulator S with inputs $y \in \mathcal{L}(\mathcal{R})$ and $c \in \mathcal{C}$ that returns protocol transcript (t, c, s), whose distribution is computationally indistinguishable from accepting protocol transcripts which have been generated by genuine protocol runs.

An example of a Σ-protocol is the Schnorr's Identification Protocol, which forms the building block of the zero-knowledge proofs used in this work and can schematically be represented as follows [19]:

Prover		**Verifier**
Input: $w \in \mathbb{Z}_q$	Common Input: $y \leftarrow g^w$	
$k \in_R \mathbb{Z}_q$ $t \leftarrow \mathsf{Com}(k) \leftarrow g^k$		
	t $\longrightarrow$	
		$c \in \{0,1\}^m$
	$\longleftarrow$ c	
$s \leftarrow k - cw \in \mathbb{Z}_q$		
	s $\longrightarrow$	
		Output: $g^s y^c \stackrel{?}{=} t$

Figure 2.4: Schnorr's Identification Protocol

Similar to the Pedersen commitment scheme, assume that the Schnorr's Identification Protocol works over group $\mathcal{G} \subseteq \mathbb{Z}_p^*$ with order q and generator g, where p and q are prime and $q|p-1$ applies. The prover has witness w as private input and claims that it knows the discrete logarithm of y, such that $y \leftarrow g^w$. Relation $\mathcal{R}$ is therefore defined as: $\mathcal{R} = \{((\mathcal{G}, q, g, y), w) \mid y \leftarrow g^w\}$.

The protocol transcript can now be explained as follows:

- First, the prover computes commitment $t = \mathsf{Com}(k)$ for a random $k \in_R \mathbb{Z}_q$ and sends t to the verifier.
- The verifier computes challenge $c \in \{0,1\}^m$, where m denotes the level of soundness, and sends c to the prover.
- Subsequently, the prover computes response $s \leftarrow k - cw \in \mathbb{Z}_q$ and sends s to the verifier.
- Finally, the verifier computes $g^s y^c$, checks if it equals t and outputs its answer (accept or reject).

2.3.2.2 Non-Interactive Zero-Knowledge Proofs

A Σ-proof with the special soundness property can be turned into a non-interactive zero-knowledge proof (NIZK) by applying the Fiat-Shamir Heuristic as follows:

Prover		**Verifier**
Input: $w \in \mathbb{Z}_q$	Common Input: $y = g^w$, $H : \{0,1\}^* \to \mathbb{Z}_q$	
$k \in_R \mathbb{Z}_q$ $t = \mathsf{Com}(k) = g^k$ $c = H(g, y, t) \in \mathbb{Z}_q$ $s = k - cw \in \mathbb{Z}_q$		
	(t, c, s) $\longrightarrow$	
		Output: $H(g, y, g^s y^c) \stackrel{?}{=} c$

Figure 2.5: Non-Interactive Schnorr's Identification Protocol

The setup for the non-interactive Schnorr's Identification Protocol is similar as in the previous section. However, the prover and verifier now also agree on a hash function $H : \{0,1\}^* \to \mathbb{Z}_q$ prior to the protocol. The prover computes t and s as in the previous section, but it now also computes challenge c as the hash function of (g, y, t) and sends (t, c, s) to the verifier. The verifier computes the hash of $(g, y, g^s y^s)$ and checks if it equals c.

According to Bernhard et al. [8], two variants of the Fiat-Shamir heuristic are commonly used: the weak variant, in which only the commitment to the prover's secret is included in the challenge and the strong variant, in which also the to be proved statement is included in the challenge. The strong variant has been proven to be secure against malicious verifiers in the random oracle model [8].

The random oracle model can be informally defined as an illusionary model in which it is assumed that a party gives an input a to a so-called oracle and receives a uniformly distributed output $R(a)$ from the oracle back, where $R : \{0,1\}^l \to \{0,1\}^t$ for lengths l and t. The output will always be equal for the same input. For the non-interactive zero-knowledge proof as depicted in Figure 2.5, we can simulate such an oracle by means of a secure cryptographic hash function: the prover gives parameters (g, y, t) to the "oracle" and receives a $H(g, y, t)$ back as challenge. The verifier can do the same with $(g, y, g^s y^c)$ and obtains the same output from the oracle, since the input is equal to the prover's input, if the protocol is secure [20].

Chapter 3

Related Work

Several approaches have been constructed in the field of BTP schemes and other biometric protocols. First of all, in Section 3.1 we explain some state-of-the-art BTP schemes that are secure in the semi-honest model. In Section 3.2, we dive deeper into biometric protocols that are secure in the malicious model.

3.1 BTP schemes in the semi-honest model

Biometric Template Protection schemes can be divided into the following categories [38, 56]:

- *Feature Transformations*: In BTP schemes based on Feature Transformations, a (one-way) transformation function is applied on both the biometric template and the database template and the two templates are compared in the transformed domain [31,56]. The revocability property is satisfied, since a template can be re-issued when it is compromised by performing a transformation function again [54]. BTP schemes based on Feature Transformations can be further divided into: BioHashing and Cancelable Biometrics. In schemes based on BioHashing, the transformation function is invertible, which means that if an attacker obtains the transformation key and the transformed template, he can retrieve the original template [38]. Cancelable Biometrics are more secure than BioHashing, since cancelable biometrics are based on non-invertible transformation functions, although the scheme still struggles with a trade-off between non-invertibility and accuracy [38, 56]. Examples of secure and efficient implementations based on Cancelable Transformations, are the works of Sadhya et al. [58], Gomez-Barrero et al. [32] and Martiri et al. [48], which proposed and implemented BTP schemes based on bloom filters. Schemes based on Bloom filters transform unprotected templates into bloom filter sets using irreversible transformation functions [32]. Bloom filter-based approaches allow for better template security, biometric data compression and efficient verification of templates [57].

- *Biometric Crypto-Systems*: In Biometric crypto-systems, templates are transformed in such a way that they resemble the original template of that user using error-correction coding techniques. Biometric crypto-systems do not satisfy non-invertibility, accuracy and revocability [56].

- *Biometric systems in the encrypted domain*: Biometric systems in the encrypted domain are systems that perform the comparison of templates in the encrypted domain in order to achieve privacy, using Homomorphic Encryption or Garbled Circuits [31, 56]. Systems in the encrypted domain satisfy non-invertibility, linkability and revocability. However, due to complex cryptographic operations, some schemes might suffer from the trade-off between accuracy and efficiency. [56]. In schemes based on garbled circuits is the comparison of two templates done using a secure multi-party protocol, called garbled circuits, in which the sensor device and the server jointly compute the result of the comparison using Boolean gates [67]. Templates based on homomorphic encryption are encrypted and compared in the encrypted domain using homomorphic operations [56]. Efficient homomorphic encryption is only very recent [31]. Examples of secure and efficient implementations of a biometric system based on homomorphic encryption are the works of Peeters et al. [56] and Gomez-Barrero et al. [31]. In the work of Peeters et al. [56], a new approach is designed and implemented that uses elliptic curve-based homomorphic encryption for verification to maintain efficiency. Moreover, it is based on a very accurate log-likelihood classifier. The work of Gomez-Barrero et al. [31] is also based on homomorphic

encryption, but is focused on multi-biometric template protection (MBTP) schemes. Multi-biometric refers to the comparison of multiple biometrics of the same subject in order to achieve a higher accuracy. The work of Gomez-Barrero et al. [31] is the first MBTP scheme that uses homomorphic encryption.

Finally, also secure and efficient systems in which new encryption and matching techniques are used, such as the work of Zhang et al. [66], which provides an approach to a cloud-based biometric authentication system. In this protocol, a database owner retrieves encrypted templates from the cloud server and performs the comparison operation. It is assumed that the cloud server is semi-honest, and that the database owner and the user can be malicious. Because of their cloud server, Zhang et al. [66] claims stronger security than similar schemes. Additionally, their scheme is secure against collusion between the client and the cloud server. However, since the cloud server is assumed to be at most semi-honest, the scheme of Zhang et al. is still not completely secure in the malicious model. We expect that extending the work of Zhang et al. [66] to the full malicious model will be harder than for other schemes, since their approach is a three-party protocol, instead of a two-party protocol.

3.2 Biometric protocols in the malicious model

The BTP schemes previously mentioned are not secure against malicious attackers, however. In the work of Bringer et al. [16], a two-party computation protocol for computing metrics (e.g. Hamming distance), called GSHADE, is extended to be secure against malicious attackers. First of all, GSHADE is extended to GSHADE$_\times$ to evaluate functions in the form $f_x(X,Y) = f_X(X) \times f_y(Y) \times \prod_{i=1}^{n} f_i(x_i, Y)$. Secondly, GSHADE$_+$, an existing protocol which is the additive variant of GSHADE$_\times$, is made secure against malicious attackers resulting in GSHADE$_\boxplus$. In GSHADE$_\boxplus$, a dual execution is performed, where two parties simultaneously execute the GSHADE$_+$ protocol, switching roles between the two executions. Subsequently, they perform a secure equality test to check whether the two protocol executions were executed correctly. Additionally, GSHADE$_\times$ is also made secure against malicious attackers by transforming GSHADE$_\boxplus$ accordingly, resulting in GSHADE$_\boxtimes$. Finally, Bringer et al. [16] concludes that the performance of GSHADE$_\boxplus$ and GSHADE$_\boxtimes$ is better than alternatives, such as cut-and-choose techniques [16]. Since GSHADE$_\boxplus$ and GSHADE$_\boxtimes$ are secure methods to compute metrics, they can be used in biometric verification systems based on e.g. the Hamming or Euclidean distance.

The partially malicious key release protocol proposed in this work differs from the work of Bringer et al. [16] in the sense that it is based on a very accurate and efficient log-likelihood ratio classifier, which is in most cases more accurate than the standard Hamming distance technique [60]. The partially malicious key release protocol, however, is only secure against a semi-honest sensor and a malicious server, as opposed to GSHADE$_\boxplus$ and GSHADE$_\boxtimes$, which are secure in the full malicious model. Nonetheless, in most cases the partially malicious key release protocol will provide sufficient security.

Chapter 4

Semi-Honest Protocol

The biometric verification protocol of Peeters et al. [56] forms the basis of this work. Hence, we will first explain the structure of the templates used in the work of Peeters et al. [56] in Section 4.1 and then explain its protocol briefly in Section 4.2.

4.1 Template Structure

A biometric template is a structure containing similarity scores which can be selected for verification purposes. A template is constructed as follows. When a user presents its biometric identifier to the enrollment party, a k-dimensional feature vector $\vec{p}$ is created. The features are quantized over 2^b bins, to limit the number of features. For each feature, a $2^b \times 2^b$ lookup table $\mathcal{T}_{b,\rho}$ is constructed containing all possible partial similarity scores $s_{x,y}$ between two users for that feature. See Equation 4.1 below.

$$\mathcal{T}_{b,\rho} = \begin{pmatrix} s_{0,0} & s_{0,1} & \cdots & s_{0,2^b-1} \\ s_{1,0} & s_{1,1} & \cdots & s_{1,2^b-1} \\ \vdots & \vdots & \ddots & \vdots \\ s_{2^b-1,0} & s_{2^b-1,1} & \cdots & s_{2^b-1,2^b-1} \end{pmatrix} \tag{4.1}$$

Partial similarity scores are added in a later stage to obtain a single similarity score, which denotes the level of affinity between two biometric identifiers. Partial similarity scores are precomputed using a log-likelihood ratio classifier. To enable encryption of partial similarity scores, they are quantized to natural numbers using uniform quantization function $q_\Delta : \mathbb{R} \to \mathbb{N}$ with step size Δ on score domain $[\mathsf{min}(s), \mathsf{max}(s)]$. We denote the total score distribution by $\mathbb{S}$ and its score domain by $[\mathsf{min}(\mathbb{S}), \mathsf{max}(\mathbb{S})] = [\sum_{i=0}^{k-1} \mathsf{min}(s_i), \sum_{i=0}^{k-1} \mathsf{max}(s_i)]$.

For each p_i, the row containing the similarity scores of p_i with other users in the corresponding lookup table is selected and included in template $\mathbf{T}_u$ as follows [56]:

$$\mathbf{T}_u = \left(\left(\mathcal{T}_{b,\rho_0}^{(p_0)}\right)^\top \mid \left(\mathcal{T}_{b,\rho_1}^{(p_1)}\right)^\top \mid \cdots \mid \left(\mathcal{T}_{b,\rho_{k-1}}^{(p_{k-1})}\right)^\top \right)^\top \tag{4.2}$$

Now $\mathbf{T}_u$ is a $2^b \times k$ matrix containing all the partial similarity scores for feature vector $\vec{p}$ with any other feature vector.

Since details on the construction of the partial similarity scores and the templates are out of scope for this research, we highly recommend interested users to read the work of Peeters et al. [56] for further specifics.

4.2 Protocol

The biometric verification protocol of Peeters et al. [56] consists of an enrollment procedure and a verification procedure. We will explain these procedures in Section 4.2.2 and 4.2.3, respectively. However, we first explain how the key setup is done, since this is an important part of the system.

4.2.1 Key Setup

For the encryption of plaintext, Peeters et al. [56] uses the ElGamal encryption system as explained in Section 2.2.1. Therefore, both the sensor device and the verification server generate their own key pair,

consisting of public key PK and secret key sk. However, in the work of Peeters et al. [56], a threshold version of ElGamal is used, which prevents parties from decrypting a ciphertext and thus leaking a template. The key setup protocol is depicted in Figure 4.1. Note that in the original semi-honest protocol by Peeters et al. [56], the key setup and (partial) decryption is done in a slightly different manner.

Sensor Device		**Server**
	$\langle g \rangle = \mathcal{G} \subseteq \mathbb{Z}_p^* \quad (p, q \in \mathbb{P}, q \mid p-1)$	
$(\mathsf{pk}_{ss}, \mathsf{sk}_{ss}) \leftarrow \mathsf{GenerateKey}()$		$(\mathsf{pk}_{sv}, \mathsf{sk}_{sv}) \leftarrow \mathsf{GenerateKey}()$
	$\mathsf{pk}_{ss} \longrightarrow$	
	$\longleftarrow \mathsf{pk}_{sv}$	
$\mathsf{pk}_{shared} \leftarrow (\mathsf{pk}_{sv})^{\mathsf{sk}_{ss}}$		$\mathsf{pk}_{shared} \leftarrow (\mathsf{pk}_{ss})^{\mathsf{sk}_{sv}}$

Figure 4.1: Adapted key setup for semi-honest protocol by Peeters et al. [56]

Similar to the Diffie-Hellman key exchange protocol, the sensor and server first agree on a common group $\mathcal{G} \subseteq \mathbb{Z}_p^*$ with generator g and prime order q for $p, q \in \mathbb{P}$ and $q|p-1$. Then they send their public keys over an established communication channel to each other. Subsequently, they both compute a shared public key by exponentiating the public key received from the other party with their own secret key.

As mentioned in Section 2.1.1, a message m, which has been double-encrypted under a shared public key pk_{shared}, is denoted by $[[m]]$. A party with secret key sk_1 can partially decrypt a ciphertext $[[m]]_{\mathsf{pk}_{shared}}$ by performing a partial decryption function D_1, which is computed as follows:

$$\mathsf{D}_1([[m]]) = \mathsf{D}_1(c_1, c_2) = (c_1^{\mathsf{sk}_1}, c_2) = (c_1', c_2') = [m] \tag{4.3}$$

As a second decryption step by the other party holding secret key sk_2, the default ElGamal decryption function, in this work denoted by D_2, can be performed as follows:

$$\begin{aligned} \mathsf{D}_2([m]) &= \mathsf{D}_2(c_1', c_2') = c_1'^{-\mathsf{sk}_2} \cdot c_2 \\ &= c_1^{-\mathsf{sk}_1 \cdot \mathsf{sk}_2} \cdot c_2 = (g^r)^{-\mathsf{sk}_1 \cdot \mathsf{sk}_2} \cdot m \cdot (\mathsf{pk}_{shared})^r \\ &= \mathsf{pk}_{shared}^{-r} \cdot m \cdot (\mathsf{pk}_{shared})^r = m \end{aligned} \tag{4.4}$$

4.2.2 Enrollment Procedure

Before the sensor device and the verification server can engage in the biometric verification protocol, a user u using the sensor device needs to be enrolled to the system first. We describe the protocol for the enrollment phase as follows.

Enrollment Party		**Server**
(1) $u, \mathbf{T}_u \leftarrow \mathsf{Capture}()$		
(2) $[[\mathbf{T}_u]] \leftarrow \mathsf{E}(\mathbf{T}_u)$		
	(3) $\xrightarrow{(u, [[\mathbf{T}_u]])}$	
		(4) $\mathsf{StoreTable}((u, [[\mathbf{T}_u]]))$

Figure 4.2: Semi-honest enrollment protocol by Peeters et al. [56]

Protocol Description:

Step 1: The enrollment party captures biometric data from the user and constructs template $\mathbf{T}_u$ as described in Section 4.1.

Step 2-3: Then, the enrollment party encrypts the template element-wise using pk_{shared} and sends $(u, [[\mathbf{T}_u]])$ to the server. However, before encryption, the enrollment party encodes the elements as exponents of generator g to enable addition of similarity scores in a later stage. We refer back to Section 2.2.2 for more information.

Step 4: The server stores the tuple in its database for later retrieval.

4.2.3 Verification Procedure

When a user has been enrolled to the system, the system can verify the user's identity by having the sensor device it uses and the verification server engage in the verification procedure, which is depicted schematically in Figure 4.3.

Sensor Device		**Server**
(5) $u, \vec{p} \leftarrow \mathsf{Capture}()$		
	(6) $\xrightarrow{\text{u}}$	
		(7) $[[\mathbf{T}_u]] \leftarrow \mathsf{FetchTemplate}(u)$
	(8) $\xleftarrow{[[\mathbf{T}_u]]}$	
(9) $[[\vec{s}]] \leftarrow \mathsf{LookUp}(\vec{p}, [[\mathbf{T}_u]])$		
(10) $[[S]] \leftarrow [[0 + \sum_{i=0}^{k-1} s_i]]$		
	(11) $\xrightarrow{[[S]]}$	
		(12) $[[\vec{C}]] \leftarrow \mathsf{Compare}([[S]], t)$
		(13) $\pi([[\vec{C}]])$
		(14) $[\vec{C}] \leftarrow \mathsf{D}_1([[\vec{C}]])$
	(15) $\xleftarrow{[\vec{C}]}$	
(16) $\vec{C} \leftarrow \mathsf{D}_2([\vec{C}])$		
(17) $\exists C_i : C_i = 0$		
$\implies S \geq t$		

Figure 4.3: Semi-honest verification protocol by Peeters et al. [56]

Protocol Description:

Step 5: In the verification procedure, a sensor device constructs a quantized probe vector $\vec{p}$ from the biometric data it captures from user u.

Step 6-8: The sensor device then makes a request for template $[[\mathbf{T}_u]]$, which the server fetches from its database.

Step 9: For each feature p_i, a partial similarity score $[[s_i]]$ is looked up in $[[\mathbf{T}_u]]$.

Step 10: Subsequently, these partial similarity scores are added in a homomorphic way to obtain $[[S]]$ as explained in Section 2.2.2. Since ElGamal is probabilistic, $[[0]]$ is added to the summation to randomize the outcome.

Step 11: Finally, the result is sent to the verification server.

Step 12: The verification server compares $[[S]]$ to a predetermined threshold t. Note that the comparison is done in the encrypted domain. Since comparison in the encrypted domain using homomorphic encryption remains impossible, a way to perform the comparison is to compute encrypted result set $[[\vec{C}]]$:

$$[[\vec{C}]] = [[\{r_i(S - t - i) \mid \forall 0 \leq i \leq \mathsf{max}(\mathbb{S}) - t, r_i \in_R \mathbb{Z}_q\}]] \tag{4.5}$$

where r_i is a blinding value which protects the results in $[[C]]$.

Step 13: To remove ordering, a permutation function $\pi([[C]])$ is performed to randomize the results in $[[C]]$.

Step 14-15: The verification server partially decrypts $[[\vec{C}]]$ using sk_{sv} and sends the resulting $[\vec{C}]$ to the sensor device.

Step 16-17: The sensor device fully decrypts the result set using sk_{ss} and checks if any of the results equals 0, in which case the sensor device concludes that the probe indeed comes from the claimed identity [56].

Chapter 5

Semi-Honest Key Release Protocol

In this chapter, we will answer the following research question:

RQ1 How can we adapt the approach by Peeters et al. [56] in such a way that a key is released upon successful verification instead of a simple "yes" or "no"?

First of all, in Section 5.1, we describe the structure of the so-called semi-honest key release protocol that we created in this chapter and finally, in Section 5.2, we elaborate on the design choices we made.

5.1 Protocol

In this section, we first present two requirements that the semi-honest key release protocol should satisfy (Section 5.1.1). Subsequently we describe how we constructed the enrollment phase (Section 5.1.3) and the verification procedure (Section 5.1.4) of the semi-honest key release protocol.

5.1.1 Requirements

Let the key that is released by the semi-honest key release protocol be denoted by K. Now we establish the following set of requirements that the semi-honest key release protocol should adhere to:

1. The verification server should not be able to learn anything about key K.
2. K should be consistent for every biometric probe $\vec{p}$ belonging to the same identity u.

5.1.2 Key Setup

The key setup is similar to the key setup of the semi-honest protocol by Peeters et al. [56] as explained in Section 4.2.1. Recall that the server and sensor first agree on a common group $\mathcal{G} \subseteq \mathbb{Z}_p^*$ with generator g and prime order q for $p, q \in \mathbb{P}$ and $q|p-1$. Subsequently, they send their public keys to each other and compute the shared public key by exponentiating the public key received from the other party with their own secret key.

5.1.3 Enrollment Procedure

First, we describe the protocol for the enrollment phase as can be seen in Figure 5.1.

Enrollment Party		**Server**
(1) $u, \mathbf{T}_u \leftarrow \mathsf{Capture}()$		
(2) $[[\mathbf{T}_u]] \leftarrow \mathsf{E}(\mathbf{T}_u)$		
(3) $K \leftarrow g^k \mid k \in_R \mathbb{Z}_q$		
(4) $[[k]] \leftarrow \mathsf{E}(k)$		
(5) $\mathsf{AES}_K(1)$		
	(6) $\xrightarrow{(u, [[\mathbf{T}_u]], ([[k]], \mathsf{AES}_K(1)))}$	
		(7) $\mathsf{StoreTable}(u, [[\mathbf{T}_u]], ([[k]], \mathsf{AES}_K(1)))$

Figure 5.1: Semi-honest enrollment protocol by Peeters et al. [56] extended with key release

In the enrollment phase, we have added the following functionalities as compared to the work of Peeters et al. [56]:

Step 3-4: The enrollment party randomly generates an integer $k \in_R \mathbb{Z}_q$, and encodes k as $K = g^k$, which is the representation of the key released upon successful verification. Subsequently, the enrollment party encrypts K using ElGamal. Recall that in this work, we use an additively homomorphic version of ElGamal, i.e. we first encode plaintext values as powers of generator g before encryption. Since $K \leftarrow g^k$, we will denote the ElGamal encryption of K by $[[k]] = (g^r, g^k \cdot h^r)$.

Step 5: The enrollment party encrypts the value 1 with key K using AES. This encryption is used later in the verification process at the sensor side to check whether it has received a valid key.

Step 7: The verification server stores $([[\mathbf{T}_u]], [[k]], \mathsf{AES}_K(1))$ under key u in a database.

5.1.4 Verification Procedure

Secondly, we describe the protocol for the verification procedure as can be seen in Figure 5.2.

Sensor Device		**Server**
(8) $u, \vec{p} \leftarrow \mathsf{Capture}()$		
	(9) $\xrightarrow{\text{u}}$	
		(10) $[[\mathbf{T}_u]] \leftarrow \mathsf{FetchTemplate}(u)$
	(11) $\xleftarrow{[[\mathbf{T}_u]]}$	
(12) $[[\vec{s}]] \leftarrow \mathsf{LookUp}(\vec{p}, [[\mathbf{T}_u]])$		
(13) $[[S]] \leftarrow [[0 + \sum_{i=0}^{k-1} s_i]]$		
	(14) $\xrightarrow{[[S]]}$	
		(15) $[[\vec{C}]] \leftarrow \mathsf{Compare}([[S]], t)$
		(16) $\pi([[\vec{C}]])$
		(17) $([[k]], \mathsf{AES}_K(1)) \leftarrow \mathsf{FetchKey}(u)$
		(18) $[[\vec{B}]] \leftarrow [[\vec{C}]] \cdot [[k]]$
		(19) $[\vec{B}] \leftarrow \mathsf{D}_1([[\vec{B}]])$
	(20) $\xleftarrow{([\vec{B}], \mathsf{AES}_K(1))}$	
(21) $\vec{B} \leftarrow \mathsf{D}_2([\vec{B}])$		
(22) $\exists B_i : \mathsf{AES\text{-}D}_{B_i}(\mathsf{AES}_K(1)) = 1$ $\implies S \geq t$		

Figure 5.2: Semi-honest verification protocol by Peeters et al. [56] extended with key release

In the verification phase, we have added the following functionalities as compared to the work of Peeters et al. [56]:

Step 17: Using identity u, we fetch $[[k]]$ from the database.

Step 18: Similar to the encryption of $[[k]]$, the plaintext elements of $[[\vec{C}]]$ are encoded as powers of g. We multiply the encrypted comparison score vector $[[\vec{C}]]$ with $[[k]]$, obtaining $[[\vec{B}]]$, which is its homomorphic addition.

Step 22: Since B_i is random for all $0 \leq i < |\vec{B}|$, the sensor is unable to know whether it is accepted to the system, as opposed to the work of Peeters et al. [56], where the user searches for a C_i which equals 1. Therefore, the sensor receives an AES encryption of the value 1 under key K. The sensor now decrypts $\mathsf{AES}_{B_i}(1)$ for all B_i. If there is a $B_i : B_i = K$, then the decryption yields 1 and the sensor knows it is accepted to the system and that B_i is a valid key.

Note that key revocation in this system is possible and straightforward, since $[[k]]$ and $\mathsf{AES}_K(1)$ are stored in the database at Step 7 and can easily be updated if the sensor device sends an update query to the verification server. Also, note that Step 10 and Step 17 could be concatenated in the implementation by fetching $[[\mathbf{T}_u]]$, $[[k]]$ and $\mathsf{AES}_K(1)$ in one query. However, we decided to maintain two database queries for clarity purposes.

5.1.5 Proof of Security, Correctness and Requirements

Although key K is random for each protocol emulation, we treat the semi-honest key release protocol as a deterministic protocol, since the output is either 0 or K and does not change in the case of multiple runs with the same biometric sample. As mentioned in Section 2.1.3.1, for proving a deterministic protocol is secure in the semi-honest model, we need both a proof of security and correctness. For the proof of correctness and security, we refer to Appendix A and B, respectively. Furthermore, see Appendix C for a proof that the semi-honest key release protocol meets the requirements as stated in Section 5.1.1.

5.2 Design Choices

In this section, we describe how we designed the system and which parameters we used for the protocols and schemes.

Communication A server should be able to maintain connections with multiple clients at the same time. However, for simplicity purposes, we implemented the protocol such that there is only one connection between a server and a client and the client resembles both the enrollment party and the sensor device.

Discrete Logarithm Group For the underlying discrete logarithm group, which is used for the ElGamal encryption system, we use elliptic curves for their performance benefits over groups that are directly based prime integers, as mentioned before in Section 2.2.3. More specifically, we have chosen the K-233 curve as selected by NIST for the efficiency benefits of Koblitz curves over other elliptic curves [41,42]. The K-233 curve provides a security level of 112 bits [5]. See Table 5.1 for an overview of the parameter selection of K-233.

Parameter	Selection
T	2
$p(t)$	$t^{233} + t^{74} + 1$
a	0
q	3450873173395281893717377931138512760570940988862252126328087024741343
G_x (polynomial basis)	172 32ba853a 7e731af1 29f22ff4 149563a4 19c26bf5 0a4c9d6e efad6126
G_y (polynomial basis)	1db 537dece8 19b7f70f 555a67c4 27a8cd9b f18aeb9b 56e0c110 56fae6a3
G_x (normal basis)	0fd e76d9dcd 26e643ac 26f1aa90 1aa12978 4b71fc07 22b2d056 14d650b3
G_y (normal basis)	064 3e317633 155c9e04 47ba8020 a3c43177 450ee036 d6335014 34cac978

Table 5.1: Parameter choices for K-233 NIST elliptic curve, where T is the normal basis type, $p(t)$ the field polynomial, a the coefficient, q the order, and (G_x, G_y) the hexadecimal coordinates of generator point G in the polynomial and normal basis, which are two different ways to represent a field [41]

Template Construction Since the focus of this work is on the security of the biometric verification system, rather than on the biometrics part as in the work of Peeters et al. [56], we decided not to compute lookup tables first and then construct a template based on these lookup tables. Alternatively, we simulate a template $\mathbf{T}$ with size $(2^b, k)$ by choosing the partial similarity scores it contains uniformly at random from the score domain $[\mathsf{min}(s), \mathsf{max}(s)]$. We have set $\mathsf{min}(s) = 0$ and $\mathsf{max}(s)$ to a random positive integer, depending on the performance tests we did (see Chapter 8). Recall that the total score distribution is denoted by $\mathbb{S}$. We have set $\mathsf{min}(\mathbb{S}) = 0$ and $\mathsf{max}(\mathbb{S}) = \mathsf{max}(s) \cdot k$, since the probe vector $\vec{p}$ in the verification phase selects k partial similarity scores from the template. We encrypt a template by encrypting its similarity scores element-wise.

We construct a probe vector by initializing it with random integers on the interval $[0, 2^b]$, so that elements of the probe vector refer to the column numbers of the features in a template which are to be selected.

AES For the enrollment procedure, we use AES-CTR with a key size of 256 bits.

Chapter 6

Partially Malicious Key Release Protocol

In this chapter, we answer the main research question:

RQ2 How can we adapt the key release protocol in such a way that it is secure in the setting where the sensor device is (semi-)honest and the server is malicious?

First of all, in Section 6.1, we explain the building blocks needed to understand some of the parts of the partially malicious key release protocol. Then, in Section 6.2 we explain how we constructed the partially malicious key release protocol, give a security proof and finally in Section 6.3, we elaborate on the design choices we made.

6.1 Building Blocks

In this section, we present the four main building blocks of the partially malicious key release protocol: The GMW compiler (Section 6.1.1), an equality of exponents proof (Section 6.1.2), the Schnorr Signature Scheme (Section 6.1.3) and a permutation proving scheme (Section 6.1.4).

6.1.1 GMW Compiler

Goldreich et al. [28, 30] proposed an approach to transform a semi-honest secure protocol into a maliciously secure protocol, called the GMW compiler. Parties run a semi-honest protocol, but due to zero-knowledge proofs they can prove they executed the protocol correctly and thus the protocol is secure against malicious attackers. In essence, parties are enforced to behave in a semi-honest way. Compiling the semi-honest secure protocol to a maliciously secure protocol can be done in polynomial time.

We will not strictly follow the GMW compiler to obtain the partially malicious key release protocol as designed in this chapter, but we will use its Augmented Coin Tossing (ACT) protocol. Moreover, we will rely on the security of the GMW compiler to come to an argumentative security proof for the partially malicious key release protocol. See Appendix E for more details.

6.1.1.1 Preliminary Functionalities

Before we explain the protocol, let us first define some preliminary functionalities needed for the GMW compiler: the Basic Coin Tossing protocol, the Authenticated Computation protocol and the Augmented Coin Tossing protocol.

Basic Coin Tossing Protocol In a coin-tossing protocol, two distrusting parties, Alice and Bob, would like to jointly compute a common bit without a trusted party. To make sure they both receive and use the same string, they can run a coin-tossing protocol. By means of commitments over their input values, they can detect cheating from the other party. Alice and Bob run the following Basic Coin Tossing (BCT) protocol [28]:

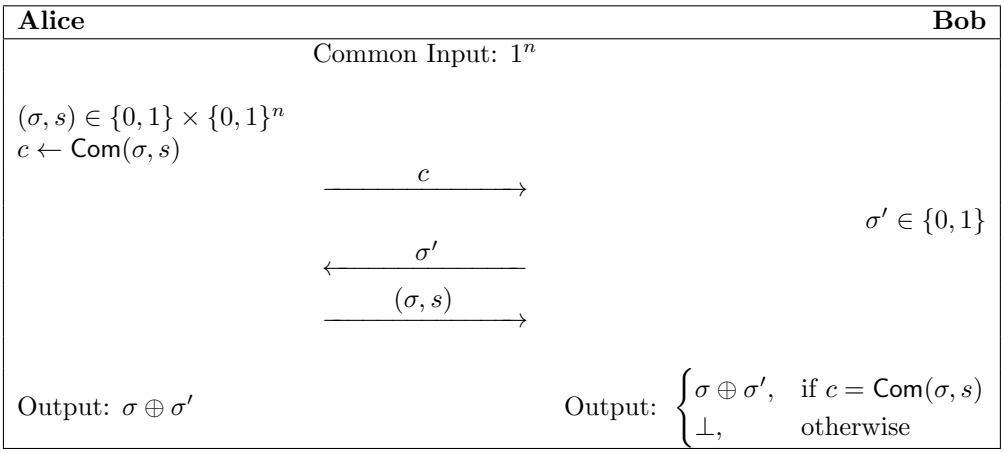


Figure 6.1: Basic Coin-Tossing Protocol

Alice and Bob enter the Basic Coin Tossing protocol with security parameter 1^n. Alice uniformly chooses bit $\sigma \in \{0,1\}$ and randomness $s \in \{0,1\}^n$ as input to commitment function Com and sends the resulting commitment c to Bob. Bob uniformly chooses bit $\sigma' \in \{0,1\}$ and sends σ' to Alice. Alice sends (σ, s) to Bob and sets her output to $\sigma \oplus \sigma'$. Bob computes $\mathsf{Com}(\sigma, s)$ and in case it equals c, Bob returns $\sigma \oplus \sigma'$. Otherwise he aborts the protocol and outputs $\perp$ [28].

Authenticated Computation Protocol The core idea of the Authenticated Computation protocol is to send the outcome of a function (e.g. commitment) on a certain input value to the other party and prove knowledge of this value in zero-knowledge, as depicted in the following figure [28]:

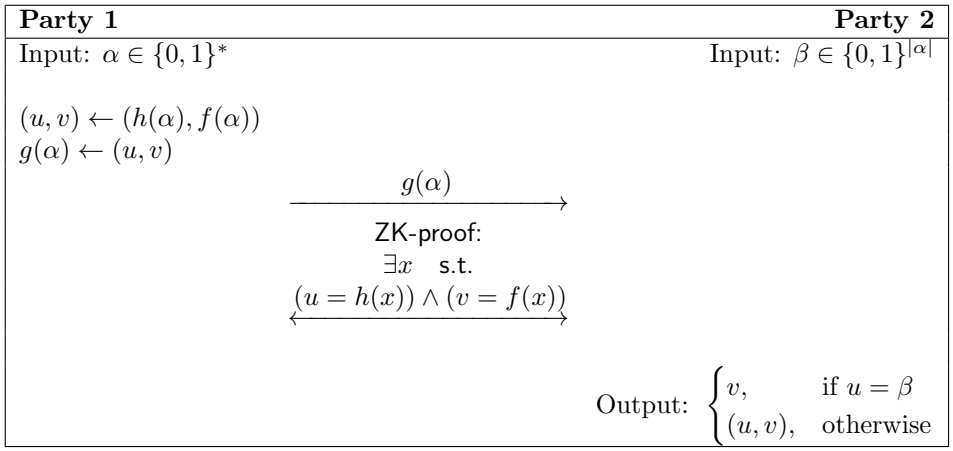


Figure 6.2: Authenticated Computation Protocol

Protocol Description:

- In the Authenticated Computation Protocol of Figure 6.2, Party 1 enters with input $\alpha \in \{0,1\}^*$ and computes function $g(\alpha) = (h(\alpha), f(\alpha))$. Party 2 enters with input $\beta \in \{0,1\}^{|\alpha|}$.
- Then, Party 1 sends $g(\alpha)$ to Party 2. Subsequently, Party 1 proofs in zero-knowledge to Party 2 that she knows x for which $u = h(x) \wedge v = f(x)$ applies. If Party 2 does not accept the proof, it aborts the protocol. Both parties enter the zero-knowledge proof with their individual computations of u and v conforming to their own views of the protocol run.
- Recall that the server might be malicious and can abort the protocol anytime. In case neither of the parties aborted the protocol in the previous steps and $u = \beta$, Party 2's output is v, which means that the protocol has succeeded. Otherwise, Party 2's output is (u, v).

Augmented Coin-Tossing The main difference with the Basic Coin Tossing protocol is that in the Augmented Coin Tossing protocol, only one party receives the random-tape and the other party receives a commitment of this random-tape. Additionally, the Basic Coin Tossing is secure in the semi-honest model and the Augmented Coin Tossing protocol is secure in the partially malicious model or the full malicious model, depending on the zero-knowledge protocols used. Since in this work we use honest-verifier zero-knowledge protocols and the sensor is always the verifier, the Augmented Coin Tossing protocol we use is secure against a semi-honest sensor and a malicious server.

The Augmented Coin-Tossing protocol is performed as depicted in Figure 6.3.

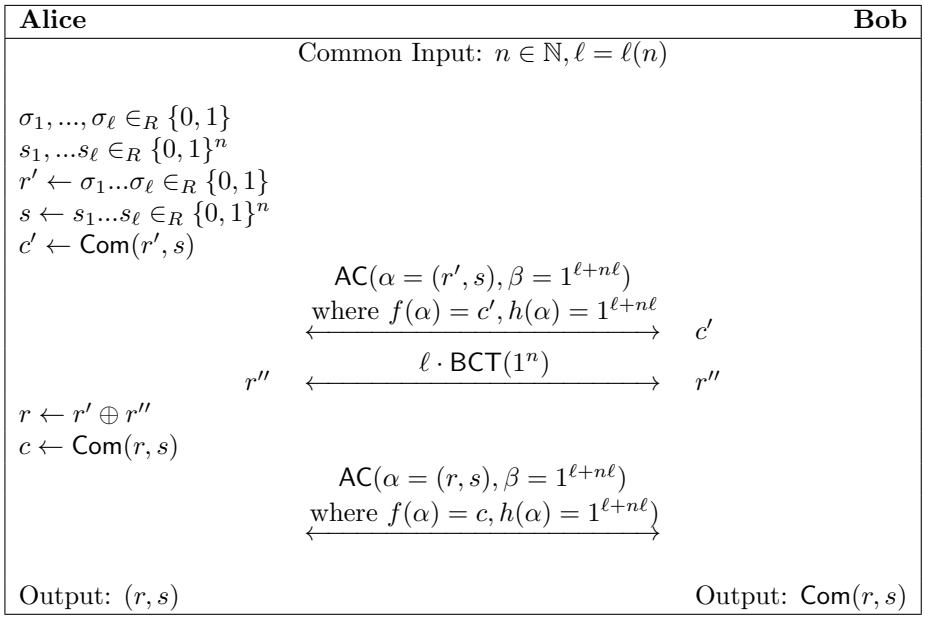


Figure 6.3: Augmented Coin-Tossing Protocol

Protocol Description:

- Both Alice and Bob obtain security parameter n and polynomial $\ell = \ell(n)$ as input.
- Alice chooses $\sigma_1, ..., \sigma_\ell \in_R \{0,1\}$ and $s_1, ...s_\ell \in_R \{0,1\}^n$ uniformly at random and sets $r' = \sigma_1...\sigma_\ell$ and $s = s_1...s_\ell$.
- Alice then computes commitment $c' = \mathsf{Com}(r', s)$ and sends c' to Bob using the Authenticated Computation Protocol where Alice enters with $\alpha = (r', s)$ and computes $f(\alpha) = c'$ and $h(\alpha) = 1^{\ell+n\ell}$. Bob enters with $\beta = 1^{\ell+n\ell}$.
- Alice and Bob engage in the Basic Coin-Tossing protocol ℓ times with security parameter 1^n as input to obtain common string $r'' \in \{0,1\}^\ell$.
- Alice computes $r = r' \oplus r''$ and $c = \mathsf{Com}(r, s)$. Again, using the Authenticated Computation Protocol, she sends c to Bob. Now, Alice enters with $\alpha = (r, s)$ and computes $f(\alpha) = c$ and $h(\alpha) = 1^{\ell+n\ell}$. Bob enters with $\beta = 1^{\ell+n\ell}$.
- Alice has now obtained random-tape (r, s) and Bob commitment $\mathsf{Com}(r, s)$ over Alice's random-tape.

6.1.1.2 Protocol

Regarding the actual GMW compiler, assume Alice and Bob have input values $x \in \{0,1\}^n$ and $y \in \{0,1\}^n$ respectively, where n is a security parameter. They would like to compute a function $f(x, y)$. Let π_{sh} be a semi-honest protocol computing $f(x, y)$. Now π_{sh} can be transformed into a protocol secure against malicious attackers using the GMW compiler as follows [28]:

Commitment of Input Values As mentioned in Section 2.1.3, a malicious attacker might participate in the protocol with an input $u' \in \{0,1\}^{|u|}$, instead of u. This substitution cannot be avoided. However, using commitment schemes and zero-knowledge proofs, we can be sure that u' is independent of the other party's input.

Alice and Bob perform the following protocol as depicted in Figure 6.4 twice. In the first invocation, Alice commits to her input value $u = x$ as Party 1 and Bob obtains the commitment to x as Party 2. In the second invocation of the protocol, the roles are reversed, with Bob committing to $u = y$ as Party 1 and Alice obtaining the commitment to y as Party 2.

Party 1		**Party 2**
Input: $u \in \{0,1\}^n$		Input: 1^n
$r' \in \{0,1\}^{n^2}$ $c' \leftarrow \mathsf{Com}(u, r')$		
	$\mathsf{AC}_1(\alpha = (u, r'), \beta = 1^{n+n^2})$ where $f(\alpha) = c', h(\alpha) = 1^{n+n^2}$ $\longrightarrow$	$y_1 = c' \vee (h(\alpha), c')$
(r, r'')	$\mathsf{ACT}(n, n^2)$ $\longrightarrow$	$c'' = \mathsf{Com}(r, r'')$
$c \leftarrow \mathsf{Com}(u, r)$		
	$\mathsf{AC}_2(\alpha = (u, r), \beta = 1^{n+n^2})$ where $f(\alpha) = c, h(\alpha) = 1^{n+n^2}$ $\longrightarrow$	$y_2 = c \vee (h(\alpha), c)$
Output: r		Output: $\begin{cases} \bot, & \text{if P1 aborts or } y_1 = (h(\alpha), c') \\ & \text{or } y_2 = (h(\alpha), c) \\ c, & \text{otherwise} \end{cases}$

Figure 6.4: Input Commitment Protocol

Protocol Description:

- First of all, Party 1 selects a uniform $r' \in \{0,1\}^{n^2}$.
- Then, Party 1 computes commitment $c' = \mathsf{Com}(u, r')$ and sends c' to Party 2 using the Authenticated Computation protocol. Party 1 enters the Authenticated Computation protocol with $\alpha = (u, r')$ and computes $f(\alpha) = c'$ and $h(\alpha) = 1^{n+n^2}$. Party 2 enters with $\beta = 1^{n+n^2}$.
- Subsequently, Party 1 and 2 invoke the Augmented Coin-Tossing protocol with input (n, n^2). Party 1 receives (r, r'') where $r \in \{0,1\}^{n^2}$ and $r'' \in \{0,1\}^{n^3}$ and Party 2 receives commitment $c'' = \mathsf{Com}(r, r'')$.
- Finally, Party 1 computes commitment $c = \mathsf{Com}(u, r)$ and sends c to Party 2 using the Authenticated Computation protocol. If Party 2 detects malicious behavior from Party 1, or if Party 1 aborts the protocol, then Party 2 outputs $\bot$ and halts the protocol. Malicious behavior occurs when Party 1 uses non-matching inputs to both invocations of the Authenticated Computation protocol, which gives $(h(\alpha), f(\alpha))$ as output.

Coin Generation A malicious party $i \in \{1,2\}$ could also be using a random-tape r_i' for the protocol that is not uniformly distributed. Therefore, Alice and Bob run the Augmented Coin-Tossing protocol twice to produce secret random-tapes r_1 and r_2, respectively, to use in the protocol emulation. The Basic Coin-Tossing protocol cannot be used here, as both Alice and Bob need their own secret random-tapes.

Through the Augmented Coin-Tossing protocol, Alice also receives $\mathsf{Decom}(c_1)$ while Bob receives $\mathsf{Decom}(c_2)$ and both parties receive commitments $\mathsf{Com}(r_1)$ and $\mathsf{Com}(r_2)$.

Protocol Emulation A malicious attacker might send other messages than prescribed by the protocol. Using the committed input and the random-tape from the previous two stages, a malicious attacker can be forced into acting semi-honest.

Using the Authenticated Computation protocol, Alice and Bob perform each step in π. Let Party j be the first party sending a message. Then the input and output pair for the Authenticated Computation protocol are made up as follows:

- $\alpha = (\alpha_1, \alpha_2, \alpha_3)$
  - $\alpha_1 = (u, \rho^j)$ where $u = x$ if $j = 1$ and $u = y$ otherwise. ρ^j denotes the randomness r used for the commitment of the input value for Party j. In other words, α_1 is the output of the Input Commitment protocol for Party j wishing to compute a commitment over his input message.
  - $\alpha_2 = (r^j, \omega^j)$. r^j and ω^j denote b and r, respectively, in the Augmented Coin-Tossing protocol. In other words, α_2 is the output that Party j received through the Augmented Coin-Tossing protocol.
  - α_3 is the sequence of messages which Party j has received earlier in the emulation of π.
- $\beta = (\gamma^j, \delta^j, \alpha_3)$
  - $\gamma^j = \mathsf{Com}(u, \rho^j)$. In other words, the first argument of β is the commitment that the second party receives over the input of Party j.
  - $\delta^j = \mathsf{Com}(r^j, \omega^j)$. In other words, the second argument of β is the commitment that the second party receives over the output of the Augmented Coin Tossing protocol of Party j.
  - α_3 is equal to a_3 above.
- Function h is defined as

$$h((v_1, s_1), (v_2, s_2), v_3) = (\mathsf{Com}(v_1, s_1), \mathsf{Com}(v_2, s_2), v_3) \tag{6.1}$$

  So, $h(\alpha) = h(\alpha_1, \alpha_2, \alpha_3) = (\mathsf{Com}(u, \rho^j), \mathsf{Com}(r^j, \omega^j), \alpha_3) = \beta$.
- Function f is the message that Party j sends in the protocol emulation.

If any aborts occurred during the protocol emulation phase, the output of both parties at the end of the protocol is $\perp$. Otherwise, the output of the parties is equal to the corresponding outputs the parties would have obtained in π_{sh}.

6.1.2 Equality of Exponents Proof

Using AND-compositions of Σ-protocols, we can prove that the discrete logarithms of two elements with different bases are equal.

Prover		**Verifier**
Input: $x \in \mathbb{Z}_q$	Common Input: $y_1 \leftarrow g^x, y_2 \leftarrow h^x$	
$k \in_R \mathbb{Z}_q$ $t_1 \leftarrow g^k$ $t_2 \leftarrow h^k$		
	$\xrightarrow{t_1, t_2}$	
		$c \in_R \{0,1\}^m$
	$\xleftarrow{c}$	
$s \leftarrow k - cx \in \mathbb{Z}_q$		
	$\xrightarrow{s}$	
		Output: $g^s y_1^c \stackrel{?}{=} t_1 \wedge h^s y_2^c \stackrel{?}{=} t_2$

Figure 6.5: Proving exponents are equal in zero-knowledge [18]

Consider the protocol as depicted in Figure 6.5. Assume a prover and verifier have agreed on a group $\mathcal{G} \subseteq \mathbb{Z}_p^*$ with generator g and order q, such that p and q are prime and $q|p-1$. The prover

has elements $y_1 = g^x$ and $y_2 = h^x$ where $x \in \mathbb{Z}_q$. Now the prover wants to prove that the discrete logarithms of y_1 and y_2 are equal. The prover chooses $k \in_R \mathbb{Z}_q$, computes $t_1 = g^k$ and $t_2 = h^k$ and sends t_1 and t_2 to the verifier. The verifier chooses challenge $c \in_R \{0,1\}^m$, where m indicates the level of soundness, and sends c to the prover. The prover computes response $s = k - cx \in \mathbb{Z}_q$ and sends s to the verifier. The verifier checks if $g^s y_1^c = t_1$ and if $h^s y_2^c = t_2$, hence the AND-composition of the Σ-protocols. The protocol can be denoted as $\mathsf{PK}\{(\alpha) : y_1 = g^\alpha \wedge y_2 = h^\alpha\}$ where PK is an abbreviation for "Proof of Knowledge" [17,18] and can be extended to $\mathsf{PK}\{(\alpha) : y_0 = g_0^\alpha \wedge ... \wedge y_i = g_i^\alpha\}$, which we will make use of in this work.

6.1.3 Schnorr Signature Scheme

The Schnorr Signature Scheme is a digital signature scheme which has been constructed by transforming the Schnorr's Identification Protocol [40] using the Fiat-Shamir heuristic in an adaptive manner. The Schnorr Signature Scheme works as follows [10]:

Similar to the Digital Signature Standard [9] and the Schnorr Identification Protocol, first a signer and a verifier agree on a group $\mathcal{G} \subseteq \mathbb{Z}_p^*$ with order q and generator g, such that p and q are prime and $q|p-1$. Moreover, the signer and verifier agree on a secure cryptographic hash function $H : \{0,1\}^* \to \mathbb{Z}_q$.

Now instead of holding a witness w and computing statement $y \leftarrow g^w$ as in the Schnorr's Identification Protocol, the signer selects $\alpha \in \mathbb{Z}_q$, computes $y \leftarrow g^\alpha \in \mathcal{G}$ and sets a secret key (p, q, g, α, H) and a public key (p, q, g, y, H).

The signer signs a message $m \in \{0,1\}^*$ as follows:

- The signer selects $k \in_R \mathbb{Z}_q$ and sets $r \leftarrow g^k \in \mathcal{G}$.
- Then it computes $c \leftarrow H(m, r) \in \mathbb{Z}_q$.
- Subsequently, the signer computes $s \leftarrow k - \alpha c \in \mathbb{Z}_q$.
- Finally, the signer returns $\mathsf{Sig}(m) \leftarrow (s, c) \in \mathbb{Z}_q^2$ as signature on message m to the verifier.

The verifier verifies message m using signature $\mathsf{Sig}(m)$ as follows:

- First, the verifier computes $v \leftarrow g^s y^c \in \mathcal{G}$.
- Then, it computes $H(m, v)$ and checks if $c \stackrel{?}{=} H(m, v)$.

The Schnorr Digital Signature Scheme is secure against existential forgery under a chosen message attack in the random oracle model, if p is chosen such that the discrete logarithm problem is hard [10,62]. A proof can be found in the works of Abdalla et al. [3] and Ohta et al. [53]. Here, we assume hash function H represents the random oracle [10].

The size of a Schnorr signature scheme is also relatively small. If we take $|p| = 1024$ bits and $|q| = 160$ bits, the signature size is only 320 bits [10] since the signature is comprised of two elements in $\mathbb{Z}_q$.

Moreover, the Schnorr Digital Signature Scheme is efficient, since the performance of an algorithm is mostly determined by the number of exponentiations and for signing, only one exponentiation is needed, which can be pre-computed [10]. For verification, two exponentiations are needed which can be performed at roughly the same cost of one exponentiation [65].

6.1.4 Permutation Proof

For proving a set of ciphertexts has been shuffled correctly, we use the zero-knowledge permutation proof in the work of Furukawa et al. [24].

Assume that a party has k ciphertexts $C = (C_1, C_1, ..., C_k)$ encrypted with ElGamal using public key (p, q, g_0, m_0, F_k) and secret key x, where g_0 generates $\mathcal{G} \subseteq \mathbb{Z}_p^*$ with order q, such that p and q are prime and $q|p-1$. Furthermore, let $m_0 = g_0^x \in \mathcal{G}$ and $F_k = \{f_v \in_R \mathcal{G}\}$ be k+5 random integers. F_k is used to satisfy the soundness property, namely that it is infeasible for any party involved in the permutation proof to generate integers $a, \{a_v\}$ where $-4 \leq v \leq k$ for which $g_0^a \prod_{v=-4}^{k} f_v^{a_v} \equiv 1 \pmod p$ applies [26].

Now we define permutation function π as:

$$\pi : \{1, ..., k\} \rightarrow \{1, ..., k\} \tag{6.2}$$

The party uses C as input to π and obtains ciphertexts $C' = (C'_1, C'_1, ..., C'_k)$ as output for which $\forall C_i : \pi(C_i) = C'_j$ applies. Permutation function π can be simulated by permutation matrix A_{ij} as follows:

$$A_{ij} = \begin{cases} 1, & \text{if } \pi(i) = j \\ 0, & \text{otherwise} \end{cases} \tag{6.3}$$

If we define ciphertexts $C_i = (g_i, m_i) \in \mathcal{G}^2$ for $1 \leq i \leq k$, then

$$C'_i = (g'_i, m'_i) = (g_0^{s_i} \prod_{j=1}^{k} g_j^{A_{ji}}, m_0^{s_i} \prod_{j=1}^{k} m_j^{A_{ji}}) \pmod p \tag{6.4}$$

where $s_i \in_R \mathbb{Z}_q$.

Now to prove that the permutation has been performed correctly, the proving and verifying party engage in the following zero-knowledge proof [24]:

$$\mathsf{PK}\{(\pi, x) : \mathsf{D}_x((g'_i, m'_i)) = \mathsf{D}_x((g_{\pi^{-1}(i)}, m_{\pi^{-1}(i)}))\} \tag{6.5}$$

for ciphertexts (g_i, m_i) where $1 \leq i \leq k$, a permutation function π, decryption function D and secret key x.

The protocol can be described as follows:

- First, the prover randomly generates $\{A_{v0}, A_{v'}\} \in_R \mathbb{Z}_q$ for $-4 \leq v \leq k$ and $\{A_{-1i}\} \in_R \mathbb{Z}_q$ for $1 \leq i \leq k$.
- Then, the prover computes the following elements:

$$A_{-2i} = \sum_{j=1}^{k} 3A_{j0}^2 A_{ji} \in \mathbb{Z}_q \quad \text{for } 1 \leq i \leq k$$

$$A_{-3i} = \sum_{j=1}^{k} 3A_{j0} A_{ji} \in \mathbb{Z}_q \quad \text{for } 1 \leq i \leq k$$

$$A_{-4i} = \sum_{j=1}^{k} 2A_{j0} A_{ji} \in \mathbb{Z}_q \quad \text{for } 1 \leq i \leq k$$

$$f'_\mu = \prod_{v=-4}^{k} f_v^{A_{v\mu}} \in \mathcal{G} \quad \text{for } 0 \leq \mu \leq k$$

$$\tilde{f}'_0 = \prod_{v=-4}^{k} f_v^{A'_v} \in \mathcal{G}$$

$$g'_0 = \prod_{v=0}^{k} g_v^{A_{v0}} \in \mathcal{G}$$

$$m'_0 = \prod_{v=0}^{k} m_v^{A_{v0}} \in \mathcal{G}$$

$$w = \sum_{j=1}^{k} A_{j0}^3 - A_{-20} - A'_{-3} \in \mathbb{Z}_q$$

$$\dot{w} = \sum_{j=1}^{k} A_{j0}^2 - A_{-40} \in \mathbb{Z}_q$$

- Subsequently, the prover sends commitment $(g'_0, m'_0, \tilde{f}'_0, \{f_{\mu'}\}, w, \dot{w})$ for $0 \leq \mu \leq k$ to the verifier.
- The verifier randomly picks challenge $c_i \in \mathbb{Z}_q$ for $1 \leq i \leq k$ and sends the challenge to the prover.

- The prover computes the following elements and sends these as response to the verifier.

  $r_v = \sum_{\mu=0}^{k} A_{v\mu} c_\mu \in \mathbb{Z}_q \quad \text{for } -4 \leq v \leq k$

  $r'_v = \sum_{i=1}^{k} A_{vi} c_i^2 + A'_v \in \mathbb{Z}_q \quad \text{for } -4 \leq v \leq k$

  where $c_0 = 1$.

- Finally, the verifier randomly chooses $\alpha \in_R \mathbb{Z}_q$. The permutation is approved if the following verifications are successful:

  $\prod_{v=-4}^{k} f_v^{r_v + \alpha r'_v} \stackrel{?}{\equiv} f'_0 \tilde{f}_0^{\prime\alpha} \prod_{i=1}^{k} f_i^{'c_i + \alpha c_i^2} \pmod p$

  $\prod_{v=0}^{k} g_v^{r_v} \stackrel{?}{\equiv} \prod_{\mu=0}^{k} g_\mu^{'c_\mu} \pmod p$

  $\prod_{v=0}^{k} m_v^{r_v} \stackrel{?}{\equiv} \prod_{\mu=0}^{k} m_\mu^{'c_\mu} \pmod p$

  $\sum_{j=1}^{k} (r_j^3 - c_j^3) \stackrel{?}{\equiv} r_{-2} + r'_{-3} + w \pmod q$

  $\sum_{j=1}^{k} (r_j^2 - c_j^2) \stackrel{?}{\equiv} r_{-4} + \dot{w} \pmod q$

Recall that k is the number of encryptions that need to be shuffled. Now, the permutation proof of Furukawa [24] requires $15k$ exponentiations and $768k$ communication bits for the prover, whereas a previous work of Furukawa et al. [26] requires $18k$ exponentiations and $1280k$ prover communication bits. Compared to other works, such as the work of Groth [33], which requires only $12k$ exponentiations and $480k$ communication bits for the prover [33], the work of Furukawa [24] is less efficient. However, the works of Furukawa et al. [24–26] are 3-move zero-knowledge protocols, while the works of Groth [33, 34] are 7-move protocols. Three-move protocols that further satisfy the requirements for a Σ-protocol as stated in Definition 10 can be used as input for the Fiat-Shamir heuristic. The Fiat-Shamir heuristic can transform an honest-verifier zero-knowledge protocol into a zero-knowledge protocol that is secure in the random oracle model.

6.2 Protocol

6.2.1 Key Setup

The key setup is similar to the setup of the semi-honest key release protocol from Section 5.1.2 and the key setup of the semi-honest protocol by Peeters et al [56]. Recall that the server and sensor first agree on a common group $\mathcal{G} \subseteq \mathbb{Z}_p^*$ with generator g and prime order q for $p, q \in \mathbb{P}$ and $q|p-1$. Subsequently, they send their public keys to each other and compute the shared public key by exponentiating the public key received from the other party with their own secret key.

We do not need an enhanced version of the key setup in the partially malicious setting. Since the sensor device is (semi-)honest, we assume that the partial similarity scores and key K are encrypted with the correct shared public key. The only case where the server needs to perform an encryption is in the comparison protocol, where the values t and $0 \leq i \leq \mathsf{max}(\mathbb{S}) - t$ need to be encrypted. However, malicious behavior of the server on the encryption part will be detected by the sensor in subsequent steps, which we will explain in the next section.

6.2.2 Enrollment and Verification Procedure

Although the enrollment and verification procedure in the semi-honest key release protocol are separately mentioned, we present the partially malicious approach as one concatenated protocol, in order to easier prove security of the complete protocol later. Since the sensor device that performs the enrollment procedure is usually different from the sensor device that engages in the actual verification procedure, we need to consider the following three-party protocol.

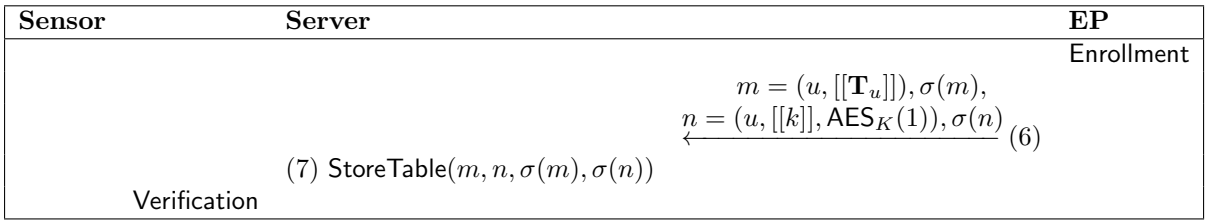


Figure 6.6: Three-party protocol

Protocol Description:

Enrollment In the three-party protocol as depicted in Figure 6.6, enrollment party (EP) first computes its enrollment, which is similar to Steps 1-5 from the semi-honest key release protocol as depicted in Figure 5.2.

Step 6 Then, the enrollment party sends the enrollment parameters $m = ([[\mathbf{T}_u]], u)$ and $n = ([[k]], \mathsf{AES}_K(1))$ along with a signature σ on m and n to the server.

Step 7 The server stores the enrollment received from the enrollment party in its database.

Verification Finally, the verification procedure between the sensor device and the server starts, which is similar to Steps 8-22 of the semi-honest key release protocol. The signature on m and n is needed, so that the server can prove later in the verification procedure that it used the correct template and keys.

For the case where the sensor device is (semi-)honest and the server malicious, we can simulate the enrollment and verification procedure as depicted in Figure 6.7. Since we consider each sensor device as (semi-)honest, the enrollment party will also be (semi-)honest and thus we simulate the protocol from Figure 6.6 in such a way that the enrollment parameters are sent from the sensor device to the server. Since the enrollment device is (semi-)honest, we will also refer to Steps 1-9 as the *trusted setup*.

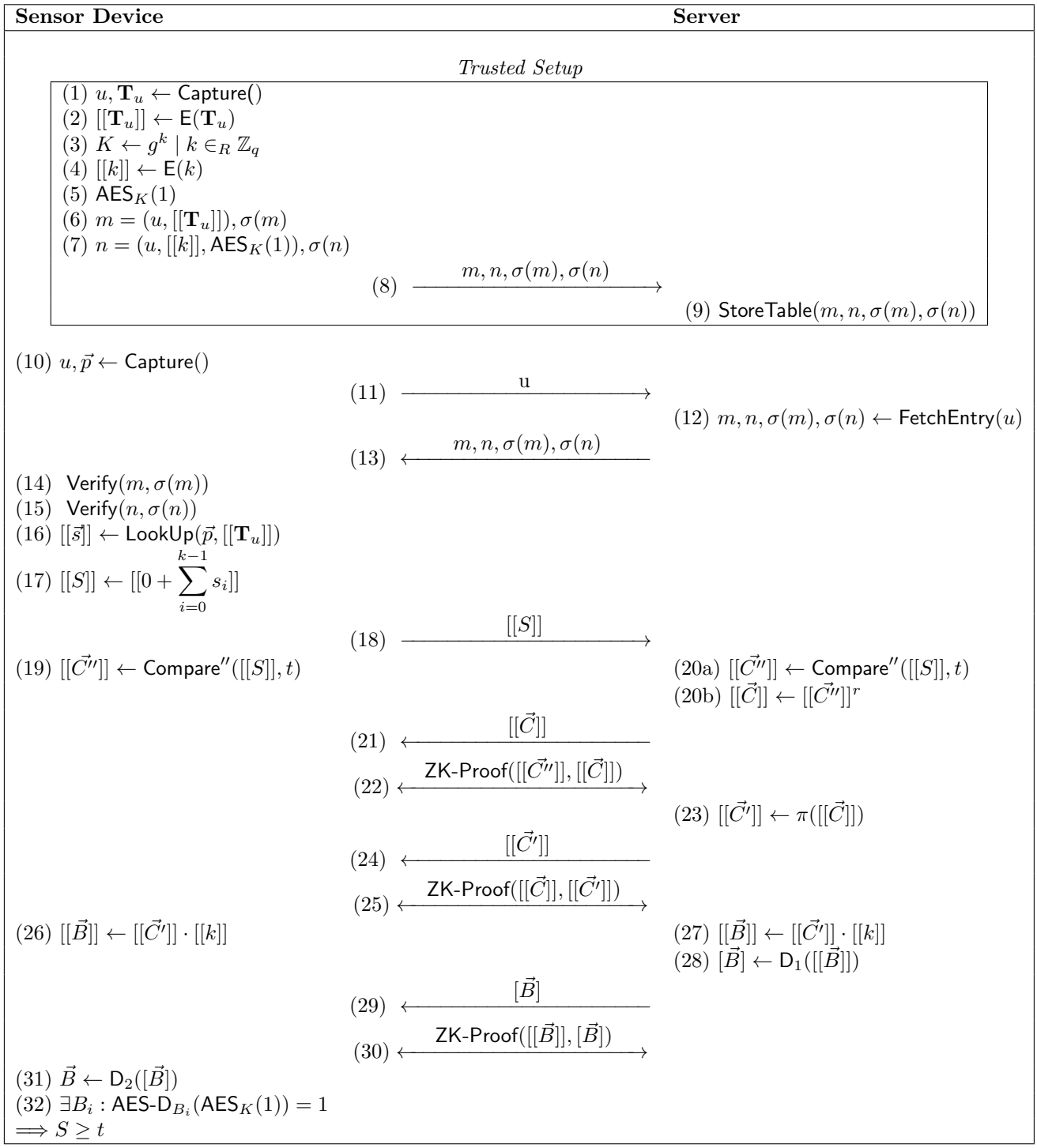


Figure 6.7: Partially malicious key release protocol

Protocol Description:

Step 1-5: Steps 1-5 are performed similar to Steps 1-5 in the semi-honest key release protocol as depicted in Figure 5.2. Since we assume the sensor device might have been compromised by only a semi-honest attacker in the worst case scenario, it does not need to prove correctness of his steps.

Step 6-7: The sensor creates signatures $\sigma(m)$ over $m = (u, [[\mathbf{T}_u]])$ and $\sigma(n)$ over $n = (u, [[k]], \mathsf{AES}_K(1))$, which are in a later step used by the server to prove that it fetched the correct entry from the database. For the signatures, we use the implementation of the Schnorr

Signature Scheme as explained in Section 6.1.3, for its aforementioned benefits. Also, since the sensor is the verifier of the signatures and is (semi-)honest, it suffices to use a signature scheme which is only honest-verifier, hence the Schnorr Signature Scheme.

Step 6-9: Steps 8 and 9 are similar to Steps 6 and 7 in the semi-honest key release protocol, with the addition that signatures $\sigma(m)$ and $\sigma(n)$ are also sent to the server and stored in the database.

Step 10-11: Steps 10 and 11 are performed similar to Steps 8 and 9 in the semi-honest key release protocol.

Step 12-13: Steps 12 and 13 are similar to Steps 10 and 11 in the semi-honest key release protocol, with the addition that signatures $\sigma(m)$ and $\sigma(n)$ are also fetched from the database and sent to the sensor.

Step 14-15: As mentioned in Step 6-7, the sensor device verifies that signatures $\sigma(m)$ and $\sigma(n)$ are valid by performing the Schnorr Signature verification procedure.

Step 16-18: Steps 16-18 are performed similar to Steps 12-14 in the semi-honest key release protocol.

Step 19-21: Recall from Equation 4.5 that $[[\vec{C}]]$ is computed as $[[\vec{C}]] = [[\{r_i(S-t-i) \mid \forall 0 \leq i \leq j, r_i \in_R \mathbb{Z}_q\}]]$ where $j = \mathsf{max}(\mathbb{S}) - t$. First of all, the server and sensor engage in the Basic Coin Tossing protocol j times to decide on random coins $x_0, ..., x_{j-1}$. Using these random coins as randomness for the ElGamal encryption of $-t-i$, they simultaneously compute $[[\vec{C''}]] = [[S-t-i]] \mid \forall 0 \leq i \leq j$ on their own. Since the randomness used for the encryption of $-t-i$ is the same for each element of $[[\vec{C''}]]$, the outcome of $[[\vec{C''}]]$ for both the server and sensor should be equal, otherwise the sensor detects cheating from the server and aborts the protocol. There is no need to verify correct computation of $[[\vec{C''}]]$ at this point, however, since malicious behavior of the server in the computation of $[[\vec{C''}]]$ will be detected by the sensor in the forthcoming zero-knowledge proof. Secondly, the server and sensor engage in the Augmented Coin Tossing protocol j times to decide on random coins $r_0, ..., r_{j-1}$ for the blinding factor which is applied to $[[S-t-i]]$. Note that the server obtains random coins $r_0, ..., r_{j-1}$ and the sensor obtains the commitments $\mathsf{Com}(r_i, \rho_i)$ to these random coins. Now the server multiplicatively blinds each element $[[\vec{C''_i}]]$ for $0 \leq i \leq j$ with random coin r_i as in Equation 4.5.

Step 22: Now, the server and sensor engage in the following zero-knowledge proof j times:

$$\mathsf{PK}\{(\alpha, \beta) : c_1 = c_1''^{\alpha} \wedge c_2 = c_2''^{\alpha} \wedge \mathsf{Com}(r_i, \rho_i) = g^{\beta} h^{\alpha}\} \tag{6.6}$$

where c_1 and c_2 denote the first and second parameter of ElGamal encryption $[[C_i]]$ and c_1'', respectively and c_2'' denote the first and second parameter of ElGamal encryption $[[C_i'']]$, respectively. $\mathsf{Com}(r_i, \rho_i)$ is a Pedersen commitment in the form $g^{\rho_i} h^{r_i}$, where we assume h is the same public key as used for ElGamal encryptions. We cannot use a commitment in the form g^r, since for the Augmented Coin Tossing protocol, the resulting commitment needs to have the hiding property.

Note that for the implementation of the zero-knowledge proof in Equation 6.6, we need the following slightly enhanced version of the equality of exponents proof from Section 6.1.2:

Prover		**Verifier**
Input: $r_i \in \mathbb{Z}_q, y_1 = c_1, y_2 = c_2, y_3 \leftarrow \mathsf{Com}(r_i, \rho_i)$		
$k_1, k_2 \in_R \mathbb{Z}_q$ $t_1 \leftarrow g^{k_1}$ $t_2 \leftarrow f^{k_1}$ $t_3 \leftarrow g^{k_2} h^{k_1}$	$\xrightarrow{t_1, t_2, t_3}$	
		$c \in_R \{0,1\}^m$
	$\xleftarrow{c}$	
$s_1 \leftarrow k_1 - cr_i \in \mathbb{Z}_q$ $s_2 \leftarrow k_2 - c\rho_i \in \mathbb{Z}_q$		
	$\xrightarrow{s_1, s_2}$	
		Output: $c_1''^{s_1} y_1^c \stackrel{?}{=} t_1 \wedge$ $c_2''^{s_1} y_2^c \stackrel{?}{=} t_2 \wedge g^{s_2} h^{s_1} y_3^c \stackrel{?}{=} t_3$

Figure 6.8: Proving exponents are equal in zero-knowledge [18]

In other words, the server and sensor use the equality of exponents proof as depicted in Figure 6.8 to prove that for each ElGamal encryption of the comparison value, the server used the same blinding value for both the first and the second ElGamal paremeter and that this blinding value is the randomness agreed upon between the sensor and server in the Augmented Coin Tossing protocol. Note that using AND-compositions, we can concatenate the zero-knowledge proofs for all r_i, hence we only need one zero-knowledge proof in total for proving correct comparison.

Step 23-25: The server performs the shuffling procedure as in Section 6.1.4 using a permutation matrix, and then it enters the following zero-knowledge proof:

$$\mathsf{PK}\{(\phi, x) : \mathsf{D}_x([[\vec{C'}]]) = \mathsf{D}_x([[\vec{C'}]]_{\phi^{-1}(i)}))\} \tag{6.7}$$

for permutation function ϕ and decryption function D with secret key x.

In other words, in Steps 23-25 the server proves that $[[\vec{C'}]] = \pi([[\vec{C}]])$ where π is a permutation function.

Similar to the verification of the signatures, since the sensor device is (semi-)honest, it suffices to use an honest-verifier zero-knowledge protocol, hence the permutation proof of Furukawa [24] is suitable. Furthermore, we chose to use the permutation proof of Furukawa [24] over other permutation proving schemes, since it is the most efficient one compared to other 3-move zero-knowledge protocols. A 3-move zero-knowledge protocol is needed for extending the current work to the full malicious protocol, as explained earlier in Section 6.1.4.

Step 26-27: The server performs Step 27 similar to Step 18 of the semi-honest key release protocol. The server does not need to prove in zero-knowledge the correctness of Step 27. Since $[[k]]$ has been sent to the sensor device in Step 13 and $[[\vec{C'}]]$ in Step 24, the sensor device can verify if $[[\vec{B}]]$ has been correctly computed by performing the multiplication of $[[\vec{C}]]$ and $[[k]]$ itself, hence Step 26. There is no need to verify correct computation of $[[\vec{B}]]$ at this point, however, since malicious behavior of the server in the computation of $[[\vec{B}]]$ will be detected by the sensor in Step 30.

Step 28-29: The server performs Step 28 and 29 similar to Step 19 and 20 of the semi-honest key release protocol. However, in Step 20 of the semi-honest key release protocol, $\mathsf{AES}_K(1)$ is also sent to the sensor device, whereas in the simplified malicious protocol, $\mathsf{AES}_K(1)$ has already been sent in Step 13.

Step 30: Recall that partial decryption function D_1 is denoted as $\mathsf{D}_1(c) = (c_1^{\mathsf{sk}_{sv}}, c_2)$. The server needs to prove that $[\vec{B}]$ is a partial decryption of $[[\vec{B}]]$ under secret key sk_{sv}. The server and sensor engage in the following zero-knowledge proof:

$$\mathsf{PK}\{(\alpha) : c_1' = c_1^{\alpha} \wedge \mathsf{pk}_{sv} = g^{\alpha} \mid \forall c' = [B_i], c = [[B_i]]\} \tag{6.8}$$

where c_1' and c_1 denote the first parameter of ElGamal encryptions c' and c, respectively.

Note that the sensor and server enter the zero-knowledge protocol with their own computation of $[[\vec{B}]]$ as obtained in Steps 26 and 27, respectively. Hence, malicious behavior of the server in Step 27 will be detected by the sensor, since in that case c, as computed by the sensor and server, do not match and thus the zero-knowledge proof will fail.

The parties enter the equality of exponents proof with arguments $x = \mathsf{sk}_{sv}, y_1 = c_1', y_2 = \mathsf{pk}_{sv}$. In other words, the sensor uses the equality of exponents proof as explained in Section 6.1.2 to verify that for each $[B_i]$, the correct base c_1 has been used. Moreover the sensor proves that the discrete logarithm of the first parameter of the ElGamal encryption is equal to the discrete logarithm of the server's public key, i.e. secret key sk_{sv}. Note that using AND-compositions, we only need one zero-knowledge proof to perform Step 30.

Step 31-32: Step 31 and 32 are performed similar to Steps 21 and 22 in the semi-honest key release protocol.

6.2.3 Proof of Security

First of all, we proof that the protocol constructed in this chapter is secure in the semi-honest model. Similar to the semi-honest key release protocol, we treat the function computed by the partially malicious key release protocol as deterministic, since the output of the protocol is either 0 or K. Accordingly we need both a proof of security and a proof of correctness. For the proof of correctness, we refer to Appendix D.

Secondly, using the GMW compiler and the aforementioned proof of semi-honest security, we prove that the partially malicious key release protocol is secure against a semi-honest sensor device and a malicious server.

For both proofs of security, we refer to Appendix E.

6.3 Design Choices

In this section, we describe how we designed the system and which parameters we used for the protocols and schemes. See also Section 5.2 for the design choices of the underlying semi-honest key release protocol.

Soundness Parameter As recommended by the ZKProof Community Reference [68], the soundness parameter (i.e. length of the challenge in bits) of a zero-knowledge proof should be set to 40, 64, 80 or 128 bits, depending on the context. It is specifically recommended to choose a soundness parameter of at least 64 bits, hence we have set the soundness parameter to 80 bits for all the zero-knowledge proofs in this work. The probability that the verifier now rejects the proof is as low as 2^{-80}.

Coin Tossing Protocols For the commitment schemes in the Basic Coin Tossing protocol and the Authenticated Computation Protocol, we use a Pedersen commitment scheme as explained in Section 2.3.1. Since the Pedersen commitment scheme is additively homomorphic [61], we will work with commitment function inputs $(\sigma, s) \in \{0,1\}^{\ell} \times \{0,1\}^{\ell \cdot n}$, instead of $(\sigma, s) \in \{0,1\} \times \{0,1\}^{n}$, to reduce the computational costs of calling the Basic Coin Tossing ℓ times in the Augmented Coin Tossing protocol.

Schnorr Signature Scheme Since the underlying group in our protocol is based on an elliptic curve and we compute signatures on elements in this group, we use the same elliptic curve for the underlying group of the Schnorr Signature Scheme. Moreover, Schnorr Signature schemes based on elliptic curves are considered very efficient [51].

For the hash function used in the Schnorr Signature Scheme, we use SHA-512, since there is no current implementation of SHA-3 algorithms available in libscapi [2] yet and SHA-512 is as secure as SHA-256 and SHA-384, but has better performance in terms of speed on 64-bit systems [35]. According to Neven et al. [51], security of a signature scheme is not dependent on the collision resistance property of a hash function and thus SHA-1/SHA-256 hash functions are secure to use in the Schnorr Signature Scheme [51]. Therefore, we assume SHA-512 to be secure in the random oracle model.

Chapter 7

Semi-Honest One-Round Protocol

In this chapter, we will answer the following research question:

RQ4 How can we transform the semi-honest key release protocol in such a way that it is a one-round protocol in which comparison is done on the server side?

In Section 7.1, we propose a semi-honest one-round protocol in which the computation of the similarity score of the probe with the database template is completely performed by the verification server. Since this chapter is not the main focus of this research, we have not implemented the semi-honest one-round key release protocol.

7.1 Protocol

The semi-honest one-round protocol consists of three phases: the key setup, the enrollment procedure and the verification procedure, which are similar to the other protocols in this work. The difference, however, lies in the verification procedure, where the complete comparison is performed by the server.

Also, for encryption, we rely on the somewhat homomorphic BV-scheme by Brakerski et al. [15], instead of ElGamal. The BV-scheme [15] provides support for both additions and (a limited number of) multiplications, whereas ElGamal is multiplicatively homomorphic by default or can be used as an additively homomorphic encryption scheme if plaintext elements are encoded as exponents of generator g. Since we need both multiplications and additions in the semi-honest one-round protocol, we use the BV-scheme [15] to encrypt plaintext in the semi-honest one-round protocol. Even though a full homomorphic encryption scheme also supports both additions and multiplications, we chose to use a somewhat homomorphic encryption scheme, for its performance benefits over full homomorphic encryption schemes [52]. Other somewhat homomorphic encryption schemes, such as the GHV-scheme by Gentry et al. [27] and the BGN-scheme by Boneh et al. [11] could be used as well in the semi-honest one-round protocol [13].

We denote the homomorphic properties of the BV-scheme for messages $m, m' \in \mathcal{M}$ as follows:

$$[m] + [m'] = [m + m'] \tag{7.1}$$

$$[m] \cdot [m'] = [m \cdot m'] \tag{7.2}$$

In this work, we will treat the BV-scheme [15] as a black box, since it is not the main focus of this research. We refer to the work of Brakerski et al. [15] for more details on the key setup, encryption and decryption in the BV-scheme [15].

7.1.1 Key Setup

The key setup for the semi-honest one-round key release protocol is rather trivial, since the sensor device only generates a key pair $(\mathsf{pk}_{ss}, \mathsf{sk}_{ss})$ and then sends public key pk_{ss} to the verification server. The server needs the public key later in the verification procedure to encrypt threshold t and $0 \leq i \leq \mathsf{max}(\mathbb{S}) - t$. We do not need a key setup similar to the previous protocols as depicted in Figure 4.1, since there is no need for threshold encryption in the current protocol. Recall that the reason for threshold encryption in the previous protocols was to prevent both the sensor and the server from having a secret key to decrypt the template or the partial similarity scores. As can be seen in Figure

7.2, although the sensor owns a secret key to decrypt the template and partial similarity scores (from generating the key pair), it does not have access to the template and partial similarity scores, since they are stored or computed at the server side. The server only has the public key for encryption, but it does not own the secret key for decrypting the template and similarity score.

7.1.2 Enrollment Procedure

Enrollment Party		**Server**
(1) $u, \mathbf{T}_u \leftarrow \mathsf{Capture}()$		
(2) $[\mathbf{T}_u] \leftarrow E(\mathbf{T}_u)$		
(3) $K \in_R \mathcal{M}$		
(4) $[K] \leftarrow \mathsf{E}(k)$		
(5) $\mathsf{AES}_K(1)$		
	(6) $\xrightarrow{(u, [\mathbf{T}_u], ([K], \mathsf{AES}_K(1)))}$	
		(7) $\mathsf{StoreTable}(u, [\mathbf{T}_u], ([K], \mathsf{AES}_K(1)))$

Figure 7.1: Enrollment procedure for semi-honest one-round protocol extended with key release

The enrollment procedure for the semi-honest one-round key release protocol is similar to the enrollment procedures of the semi-honest key release protocol as depicted in Figure 5.1 and the partially malicious key release protocol as depicted in Figure 6.7. However, note that the encryption of $\mathbf{T}_u$ and K is not done using a threshold encryption system, but using a secret key owned by the enrollment party, as explained in the previous section. Also, since we use the BV-scheme [15] which supports additions, we do not encode plaintext values as exponents of generator g anymore, as in ElGamal. Hence, we choose key K directly from message space $\mathcal{M}$.

7.1.3 Verification Procedure

Sensor Device		**Server**
(8) $u, \mathbf{P} \leftarrow \mathsf{Capture}()$		
(9) $[\mathbf{P}] \leftarrow E(\mathbf{P})$		
	(10) $\xrightarrow{(u, [\mathbf{P}])}$	
		(11) $[\mathbf{T}_u] \leftarrow \mathsf{FetchTemplate}(u)$
		(12) $[\mathbf{Q}] \leftarrow \mathsf{LookUp}([\mathbf{P}], [\mathbf{T}_u])$
		(13) $[S] \leftarrow [0] + \sum_{j=0}^{2^b-1} \sum_{i=0}^{k-1} [\mathbf{q}_{j,i}]$
		(14) $[\vec{C}] \leftarrow \mathsf{Compare}([S], t)$
		(15) $\pi([\vec{C}])$
		(16) $([K], \mathsf{AES}_K(1)) \leftarrow \mathsf{FetchKey}(u)$
		(17) $[\vec{B}] \leftarrow [\vec{C}] + [K]$
	(18) $\xleftarrow{([\vec{B}], \mathsf{AES}_K(1))}$	
(19) $\vec{B} \leftarrow D([\vec{B}])$		
(20) $\exists B_i : \mathsf{AES\text{-}D}_{B_i}(\mathsf{AES}_K(1)) = 1$ $\Longrightarrow S \geq t$		

Figure 7.2: Verification procedure for semi-honest one-round protocol

Protocol Description:

Step 8: As compared to the semi-honest key release verification protocol (see Figure 5.2), we replace feature vector $\vec{p}$ with matrix $\mathbf{P}$, which contains zeros on all positions by default. Recall that for

simplicity reasons, in the semi-honest (key release) protocol, $\vec{p}$ is a vector containing the column numbers of $\mathbf{T}_u$ that have been selected. In the current setting, $\mathbf{P}$ is a $2^b \times k$ bit matrix. As mentioned in Section 4.1, k is the number of lookup tables that the template contains and 2^b the size of each row that is selected from these lookup tables [56]. For each partial similarity score $s_{x,y} \in \mathbf{T}_u$ that should be selected for the computation of the total similarity score, the bit on position (x, y) in $\mathbf{P}$ is set.

Step 9: Matrix $\mathbf{P}$ is encrypted using the BV-scheme [15], similar to K and $\mathbf{T}_u$ in the enrollment procedure.

Step 10-11: The sensor device sends both u and the encrypted $[\mathbf{P}]$ to the verification server.

Step 12: The verification server will now do the look up procedure as opposed to the verification procedure of the semi-honest key release protocol where the sensor device performs the look up. The LookUp function is performed by multiplying $[\mathbf{P}]$ and $[\mathbf{T}_u]$ element-wise. The result of this multiplication is encrypted matrix $[\mathbf{Q}]$.

Step 13: The total similarity score $[S]$ is computed by homomorphically adding up all the elements $[\mathbf{q}_{j,i}]$ from $[\mathbf{Q}]$.

Step 14-20: The last steps are similar to Steps 15-22 of the semi-honest key release protocol. However, observe that the partial decryption step has been removed, since there is no threshold encryption involved in the current protocol.

Also note that the only multiplication we need in the semi-honest one-round key release protocol is the multiplication of $[\mathbf{P}]$ and $[\mathbf{T}_u]$ in Step 12. Hence, we claim that the semi-honest one-round protocol is efficient compared to other biometric verification protocols that use a somewhat homomorphic encryption scheme.

Chapter 8

Performance Evaluation

In this chapter, we will answer the following research question:

RQ4 How efficient is the partially malicious key release protocol and how does the partially malicious key release protocol compare to related semi-honest and malicious BTP schemes in terms of efficiency?

First, in Section 8.1 we give an overview of the parameters, scheme instantiations and the labels for the different protocol components we have set for the implementation of the semi-honest and partially malicious key release protocol. Then in Section 8.2, we measure the efficiency of the partially malicious key release protocol in terms of computational and communication complexity. Subsequently, in Section 8.4 we do a similar measure, but in terms of runtime and finally in Section 8.5 we discuss the results of both assessments.

8.1 Overview of Parameters, Scheme Instantiations and Protocol Components

Although we have already mentioned the parameters, encryption schemes and proofs instantiations for the semi-honest and partially malicious key release protocol throughout this paper before, we put them together in Table 8.1 for clarity reasons.

Scheme/parameter	**Instantiation**
Encryption scheme	ElGamal
Underlying discrete log group	NIST K-233 elliptic curve (112-bits security)
Commitment scheme for AC and ACT protocols	Pedersen commitment scheme
Soundness parameter for zero-knowledge proofs	80 bits
Signature scheme	Schnorr Signature Scheme
Hash function in signature scheme	SHA-512
AES	CTR chaining mode with 256 bits key length

Table 8.1: Overview of parameters and scheme instantiations used in the implementation of the semi-honest and partially malicious key release protocol

For the parameter selection of the K-233 NIST elliptic curve, we reproduce Table 5.1 for clarity purposes below.

Parameter	Selection
T	2
$p(t)$	$t^{233} + t^{74} + 1$
a	0
q	3450873173395281893717377931138512760570940988862252126328087024741343
G_x (polynomial basis)	172 32ba853a 7e731af1 29f22ff4 149563a4 19c26bf5 0a4c9d6e efad6126
G_y (polynomial basis)	1db 537dece8 19b7f70f 555a67c4 27a8cd9b f18aeb9b 56e0c110 56fae6a3
G_x (normal basis)	0fd e76d9dcd 26e643ac 26f1aa90 1aa12978 4b71fc07 22b2d056 14d650b3
G_y (normal basis)	064 3e317633 155c9e04 47ba8020 a3c43177 450ee036 d6335014 34cac978

Table 5.1: Parameter choices for K-233 NIST elliptic curve, where T is the normal basis type, $p(t)$ the field polynomial, a the coefficient, q the order, and (G_x, G_y) the hexadecimal coordinates of generator point G in the polynomial and normal basis, which are two different ways to represent a field [41]

In this chapter, we measure the time complexity and the runtime of the following components of the partially malicious key release protocol:

Protocol component	Description
Setup	Includes the generation of the ElGamal keys and the set up of the threshold system. For the experimental analysis, we take the fastest computation as the setup is done simultaneously.
Enrollment	Step 1-7 and Step 9
Capture	Step 10
Fetch from table	Step 12
Signature verification	Step 14 and 15.
Look up	Step 16
Addition of scores	Step 17
Comparison	Step 19 or 20a (for the experimental analysis we take the fastest computation as the comparisons are done simultaneously) and Step 20b.
ZK-Proof comparison	Step 22.
Permutation	Step 23.
ZK-Proof permutation	Step 25.
Addition of key K	Step 26 or 27. Again, for the experimental analysis we take the fastest computation as the additions are done simultaneously.
Partial decryption	Step 28.
ZK-Proof partial decryption	Step 30.
Full decryption	Step 31.
Key check	Step 32.
Total communication overhead	Steps 8, 11, 13, 18, 21, 24 and 29. For simplicity reasons, the communication overhead produced by the zero-knowledge proofs is already included in the zero-knowledge proofs themselves.

Table 8.2: Overview of components in the partially malicious key release protocol and which steps in the protocol they contain

8.2 Computational Complexity

To measure the efficiency of the partially malicious key release protocol, we express the number of operations of the partially malicious key release protocol in terms of elliptic curve scalar point multiplications (M) and point additions (A).

8.2.1 Building Blocks

First of all, we describe the number of multiplications (M) and additions (A) required for several building blocks of the partially malicious key release protocol.

We define the following functions for the time complexity of an ElGamal setup procedure ($T_{EG_{setup}}$), an additively homomorphic ElGamal encryption operation ($T_{EG_{enc}}$), an ElGamal decryption operation ($T_{EG_{dec}}$), an ElGamal ciphertext multiplication ($T_{EG_{mult}}$), an ElGamal ciphertext exponentiation ($T_{EG_{exp}}$), a Pedersen commitment computation ($T_{Com_{Pedersen}}$), a computation of a Schnorr signature ($T_{Schnorr_{sig}}$) and a computation of a Schnorr verification ($T_{Schnorr_{ver}}$):

$$T_{EG_{setup}} = M \tag{8.1}$$

$$T_{EG_{enc}} = 3M + A \tag{8.2}$$

$$T_{EG_{dec}} = M + A \tag{8.3}$$

$$T_{EG_{mult}} = 2A \tag{8.4}$$

$$T_{EG_{exp}} = 2M \tag{8.5}$$

$$T_{Com_{Pedersen}} = 2M + A \tag{8.6}$$

$$T_{Schnorr_{sig}} = M + A \tag{8.7}$$

$$T_{Schnorr_{ver}} = 2M + A \tag{8.8}$$

An invocation of the Basic Coin Tossing protocol includes 2 commitments. Since we use Pedersen commitments in this work, we define the computational complexity of the Basic Coin Tossing protocol as:

$$T_{BCT} = 2T_{Com_{Pedersen}} = 4M + 2A \tag{8.9}$$

The Augmented Coin Tossing protocol consists of 2 commitments, 1 invocation of the Basic Coin Tossing protocol and 2 invocations of the Authenticated Computation protocol. Since we use Pedersen commitments and Σ-protocols, we can express the cost of a single invocation of the AC Protocol for the ACT protocol as $T_{AC} = 5M + 3A$. We now express the cost of the ACT protocol as:

$$\begin{aligned} T_{ACT} &= 2T_{Com_{Pedersen}} + T_{BCT} + 10M + 6A \\ &= 18M + 10A \end{aligned} \tag{8.10}$$

8.2.2 Protocol Components

Using the computational costs of the building blocks, we can describe the complexities of the components of the partially malicious key release protocol as defined in Table 8.2 as follows:

Setup: Both the server and sensor generate their own public key (2 scalar point multiplications) and upon receiving the public key of the other party they exponentiate the public key received from the other party by their own secret key (2 scalar point multiplications) to obtain the shared public key. However, since the computation of the individual and shared public keys occurs simultaneously, for the computational cost of the setup, we only include the key computation of one party, hence:

$$T_{Setup} = 2M \tag{8.11}$$

(1-7 and 9) Enrollment: Since templates are encrypted element-wise, we need $k2^b$ ElGamal encryptions in Step 2. Since an additively homomorphic EC-ElGamal encryption requires 3 scalar point multiplications and 1 point addition (see Equation 8.2), the cost for Step 2 can be expressed as $T_{Step_2} = k2^b \cdot T_{EC_{enc}}$. Step 3-4 requires 1 additively homomorphic ElGamal encryption, i.e. $T_{Step_{3-4}} = T_{EC_{enc}}$. In Step 6 and 7 the enrollment party signs 2 Schnorr signatures, i.e. $T_{Step_{6-7}} = 2T_{Schnorr_{sig}}$. So, we define the total cost for the enrollment as:

$$T_{Enroll} = (k2^b + 1)T_{EC_{enc}} + 2T_{Schnorr_{sig}} = (k2^b + 6)M + (k2^b + 3)A \tag{8.12}$$

(14-15) Signature Verification: We invoke the verification algorithm of the Schnorr Signature Scheme, twice, so we express the cost of the signature verification step as:

$$T_{SigVer} = 2T_{Schnorr_{ver}} = 4M + 2A \tag{8.13}$$

(16) Look Up: No scalar point multiplications and point additions are performed in this step.

(17) Addition of scores: For the addition of the partial similarity scores, we need to multiply $k-1$ ElGamal ciphertexts with an encryption of 0, so in total we need k ElGamal ciphertext multiplications:

$$T_{Add_{scores}} = k \cdot T_{EG_{mult}} = 2k \cdot A \tag{8.14}$$

(19-20) Comparison: Similar to the work of Peeters et al. [56], we denote the number of comparisons by $\alpha = \mathsf{max}(\mathbb{S}) - t$. For a single score comparison, first the server and sensor invoke the Basic Coin Tossing protocol to agree on a random coin used for the encryption of $-t - i$. Then, since the server and sensor encrypt $-t - i$ simultaneously, only 1 ElGamal encryption of $-t - i$ and 1 ElGamal ciphertext multiplication (point addition) for the homomorphic addition of $[[S]]$ and $[[t - i]]$ is needed for the computation of the comparison cost. For Step 20b, the application of the blinding factor, the server and sensor first engage in the Augmented Coin Tossing protocol. Subsequently, the server performs an ElGamal ciphertext exponentiation. We can now define the total cost of the comparison procedure as follows:

$$\begin{aligned} T_{Comparison} &= \alpha(T_{BCT} + T_{EG_{enc}} + A + T_{ACT} + T_{EG_{exp}}) \\ &= \alpha(27M + 14A) \end{aligned} \tag{8.15}$$

(22) ZK-Proof Comparison: A single invocation of the adapted equality of exponents proof from Figure 6.8 requires 11 scalar point multiplications and 5 point additions, so:

$$T_{ZK-Proof_{Comparison}} = \alpha(11M + 5A) \tag{8.16}$$

(23) Permutation: Recall from Equation 6.4 that the shuffling of α ciphertexts is performed as: $C'_i = (g'_i, m'_i) = (g_0^{s_i} \prod_{j=1}^{\alpha} g_j^{A_{ji}}, m_0^{s_i} \prod_{j=1}^{\alpha} m_j^{A_{ji}}) \pmod p$, where $1 \leq i \leq \alpha$. Since permutation matrix A_{ij} contains zeros and ones, we do not perform the exponentiations of $g_j^{A_{ji}}$ and $m_j^{A_{ji}}$, but we loop through the matrix and take the respective values of g^j and m^j for which the matrix contains a zero on the corresponding positions. Therefore, we only need 2α scalar point multiplications and 2α point additions:

$$T_{Permutation} = 2\alpha(M + A) \tag{8.17}$$

(25) ZK-Proof of Permutation: According to Furukawa et al. [24], the most optimal implementation of the permutation proof requires 15α ciphertext exponentiations. For the ciphertext multiplications, we assume the worst case scenario, as we do not have investigated optimization methods for the corresponding operations in the permutation proof. For the prover side, $2(\alpha + 4) + 2\alpha = 4\alpha + 8$ point additions are needed and $(\alpha + 4) + 1 + (\alpha - 1) + 2\alpha + 2\alpha = 6\alpha + 4$ point additions for the verifier side, hence:

$$T_{ZK-Proof_{permutation}} = 15\alpha M + (10\alpha + 12)A \tag{8.18}$$

(26/27) Addition of key K: For a single multiplication of an element of $[[\vec{C'}]]$ with $[[k]]$, α invocations of the Basic Coin Tossing protocol and α ElGamal ciphertext multiplications are needed, since Step 26 and 26 are performed simultaneously by the server and the sensor. Hence, in total, the cost can be expressed as:

$$T_{Add_K} = \alpha(T_{BCT} + T_{EG_{mult}}) = 4\alpha(M + A) \tag{8.19}$$

(28) Partial decryption: Recall from Equation 4.3 that a single partial decryption requires 1 integer exponentiation, so in total α scalar point multiplications are needed:

$$T_{D_1} = \alpha M \tag{8.20}$$

(30) ZK-Proof partial decryption: Recall from Equation 6.8 that the server and sensor engage in the following zero-knowledge proof: $\mathsf{PK}\{(\alpha) : c'_1 = c_1^{\alpha} \wedge \mathsf{pk}_{sv} = g^{\alpha} \mid \forall c' = [B_i], c = [[B_i]]\}$. A single invocation of the equality of exponents proof of Figure 6.5 for the current proof requires $\alpha + 1$ scalar point multiplications at the prover side, and $2(\alpha + 1)$ scalar point multiplications and 2α point additions at the verifier side, so:

$$T_{ZK-Proof_{D_1}} = 3(\alpha + 1)M + 2\alpha A \tag{8.21}$$

(32) Full decryption: We need α decryptions, so:

$$T_{D_2} = \alpha \cdot T_{EG_{dec}} = \alpha(M + A) \tag{8.22}$$

(32) Key check: No scalar point multiplications and point additions are performed in this step.

Total: We can conclude that the total computation complexity of the partially malicious key release protocol can be expressed as:

$$\begin{aligned} T_{PMKR} &= T_{Setup} + T_{Enroll} + T_{SigVer} + T_{Add_{scores}} + T_{Comparison} + T_{ZK-Proof_{Comparison}} \\ &\quad + T_{Permutation} + T_{ZK-Proof_{Permutation}} + T_{add_K} + T_{D_1} + T_{ZK-Proof_{D_1}} + T_{D_2} \\ &= (2^b k + 64\alpha + 15)M + ((2^b + 2)k + 37\alpha + 17)A \end{aligned} \tag{8.23}$$

As mentioned in Section 2.2.3, scalar point multiplications can be done using the double-and-add method. Since we work with elliptic curve K-233, we can express the cost of a single scalar point multiplication as $233(A + D)$ in the worst-case scenario, where D denotes the number of point doublings [63]. However, more efficient methods exist to perform scalar point multiplications, but we will not go into further detail. For more specifics, we refer to the works of Aranha et al. [4], Lim et al. [44] and Bartolini et al. [6],

8.3 Communication Complexity

We also take note of the communication complexity of the partially malicious key release protocol, which we will express in the number of bits required for the message transfers. Since we work with elliptic curves, we denote the size of a curve element as $C_{element}$ and the size of an ElGamal ciphertext as $C_{EG_{ciph}} = 2C_{element}$. In case a value is an element of $\mathbb{Z}_q$, we denote its size by $|q|$. The size of $\mathsf{AES}_K(1)$ depends on the chaining mode used in AES, hence we will leave the notation of its size as $|\mathsf{AES}_K(1)|$. Furthermore, recall that u is the identity claim a sensor device makes, $(2^b, k)$ the size of a biometric template, α the number of comparisons a server performs, and m the soundness parameter in the zero-knowledge proofs.

First of all, for the enrollment phase, we can express the communication complexity as:

$$\begin{aligned} C_{enroll} &= |m| + |n| + |\sigma(m)| + |\sigma(n)| \\ &= |u| + k \cdot 2^b \cdot C_{EG_{ciph}} + |u| + C_{EG_{ciph}} + |\mathsf{AES}_K(1)| + |q| + |q| \\ &= 2|u| + (k \cdot 2^b + 1)C_{EG_{ciph}} + |\mathsf{AES}_K(1)| + 2|q| \end{aligned} \tag{8.24}$$

Analogously, we can express the communication complexity of the verification phase as:

$$|[[S]]| = C_{EG_{ciph}} \tag{8.25}$$

$$|[[\vec{C}]]| = \alpha \cdot C_{EG_{ciph}} \tag{8.26}$$

$$|\mathsf{ZK\text{-}Proof}([[\vec{C''}]], [[\vec{C}]])| = \alpha(3C_{EG_{ciph}} + m + 2|q|) \tag{8.27}$$

$$|[[\vec{C'}]]| = \alpha \cdot C_{EG_{ciph}} \tag{8.28}$$

$$\begin{aligned} |ZK - Proof([[\vec{C}]], [[\vec{C'}]])| &= 3C_{EG_{ciph}} + 2|q| + (\alpha+1)C_{EG_{ciph}} + \alpha|q| + 2|q| \\ &= (\alpha+4)(C_{EG_{ciph}} + |q|) \end{aligned} \tag{8.29}$$

$$|[\vec{B}]| = \alpha \cdot C_{EG_{ciph}} \tag{8.30}$$

$$|\mathsf{ZK\text{-}Proof}([[\vec{B}]], [\vec{B}])| = (\alpha+1)C_{EG_{ciph}} + m + |q| \tag{8.31}$$

$$\begin{aligned} C_{verification} &= |u| + |m| + |n| + |\sigma(m)| + |\sigma(n)| + |[[S]]| + |[[\vec{C}]]| + |\mathsf{ZK\text{-}Proof}([[\vec{C''}]], [[\vec{C}]])| \\ &+ |[[\vec{C'}]]| + |\mathsf{ZK\text{-}Proof}([[\vec{C}]], [[\vec{C'}]])| + |[\vec{B}]| + |\mathsf{ZK\text{-}Proof}([[\vec{B}]], [\vec{B}])| \\ &= |u| + 2|u| + (k \cdot 2^b + 1)C_{EG_{ciph}} + |\mathsf{AES}_K(1)| + 2|q| + C_{EG_{ciph}} \\ &+ \alpha \cdot C_{EG_{ciph}} + \alpha(3C_{EG_{ciph}} + m + 2|q|) + \alpha \cdot C_{EG_{ciph}} \\ &+ (\alpha+4)(C_{EG_{ciph}} + |q|) + \alpha \cdot C_{EG_{ciph}} + (\alpha+1)C_{EG_{ciph}} + m + |q| \\ &= 3|u| + |\mathsf{AES}_K(1)| + (8\alpha + 7 + k \cdot 2^b)C_{EG_{ciph}} + (\alpha+1)m + (3\alpha+7)|q| \end{aligned} \tag{8.32}$$

In our implementation, we use the K-233 elliptic curve, hence, the size of an element $C_{element} = 233$ bits and $|q| = 233$ bits. Furthermore, we have used AES-CTR, hence $|\mathsf{AES}_K(1)| = 32$ bits and we have set the soundness parameter $m = 80$. Therefore, in the current context, we can express the communication complexity (in bits) of the enrollment and verification phases of the partially malicious key release protocol as:

$$\begin{aligned} C_{enroll} &= 2|u| + (k \cdot 2^b + 1)466 + 32 + 466 \\ &= 2|u| + 466k \cdot 2^b + 958 \end{aligned} \tag{8.33}$$

$$\begin{aligned} C_{verification} &= 3|u| + 32 + (8\alpha + 7 + k \cdot 2^b)466 + (\alpha+1)80 + (3\alpha+7)233 \\ &= 3|u| + 4507\alpha + 466k \cdot 2^b + 5005 \end{aligned} \tag{8.34}$$

8.4 Experimental Analysis

For the purpose of an experimental performance analysis of the semi-honest and partially malicious key release, we have implemented proof-of-concepts for both protocols using the system setup as described in Section 8.4.1. In Section 8.4.2, we describe the different parameter sets and the results we obtained from running the experiments.

8.4.1 System Setup

The runtime experiments have been conducted on a Windows 10 (64 bit) system with an Intel Core i7-7700HQ CPU @ 2.80 GHz processor and 16.0 GB RAM, running an Ubuntu 16.04 subsystem. For the implementation of all the protocols in this work, we use the C++ version of libscapi [2], which can be found on `https://github.com/cryptobiu/libscapi`. Libscapi [2] provides a range of cryptographic primitives and tools, such as an implementation of the ElGamal cryptosystem, commitment schemes and zero-knowledge proofs, which are used in this work. The code of the protocol implementations in this work is available on `https://github.com/Ruth1993/thesis`. For instructions on how to install the code and its dependencies, we refer to Appendix G.

On the current system, measured over 1000 samples, an encryption using elliptic curve K-233 takes on average about 450 μs, a decryption takes about 230 μs, a homomorphic addition (ciphertext multiplication) about 545 μs and a scalar multiplication (ciphertext exponentiation) about 460 μs. Note that a scalar multiplication is relatively fast in the current context, since taking the components of the ciphertext as elliptic curve points manually and exponentiating these with the corresponding scalar factor appears to be faster than the default ElGamal exponentiation functionality in the implementation of libscapi [2].

8.4.2 Results

Since there are several parameters which determine the total runtime of the semi-honest and partially malicious key release protocol, we need to define sets of realistic parameters, i.e. feature sets for the runtime assessment. First of all, according to Peeters et al. [56], the total runtime is dominated by the comparison operation, which requires $\alpha = \mathsf{max}(\mathbb{S}) - t$ score comparisons. Similar to the work of Peeters et al. [56], for the semi-honest key release protocol, a complete single score comparison includes one homomorphic addition $[[S - t - i]]$, one scalar multiplication $[[r(S - t - i)]]$, a partial decryption $[r(S-t-i)]$ and a full decryption $S-t-i$. However, since we have added a key release in this work, we also include the homomorphic addition of the key $[[S-t-i+k]]$. Moreover, we include the permutation function. Although a single comparison result cannot be permuted, we still add the permutation function to α, since we measure α over multiple comparisons. For the partially malicious key release protocol, we also include the zero-knowledge proofs for the comparison, permutation and partial decryption and the communication overhead the zero-knowledge proofs require by sending $[[\vec{C}]], [[\vec{C'}]]$ and $[\vec{B}]$ before the proofs start.

According to Peeters et al. [56], $\mathsf{max}(\mathbb{S})$ depends on the number of features k, the quality of the feature ρ, the feature quantization parameter b and the step size Δ for the score quantization function. Since the focus of this work is not on the biometric part, but rather on the encryption part, we do not use real or simulated biometric data, but we randomly generate templates and probe vectors. As a consequence, we disregard the effect of the quality of the feature ρ on $\mathsf{max}(\mathbb{S})$, since in the current work this parameter is not included. Instead, we include the maximum score for a single partial similarity score $\mathsf{max}(s)$ as parameter influencing $\mathsf{max}(\mathbb{S})$, since in this work $\mathsf{max}(\mathbb{S})$ is computed as $\mathsf{max}(\mathbb{S}) = k \cdot \mathsf{max}(s)$. From the work of Peeters et al. [56], we have deducted that $\mathsf{max}(s) = 6$, hence we will use the same value for the runtime assessment of the partially malicious key release protocol. Note that since $\mathsf{max}(\mathbb{S}) = k \cdot \mathsf{max}(s)$ in this work, feature quantization parameter b does not have any effect on $\mathsf{max}(\mathbb{S})$. However, as b does influence other parts of the protocol, such as the enrollment, we still include b as parameter for the runtime assessment.

Threshold t depends on the FAR and FRR. There are several measures to choose t based on the FAR and FRR [47], but in the work of Peeters et al. [56], t is chosen such that the FAR and FRR are equal, i.e. there is a trade-off between the number of falsely accepted probe vectors and falsely declined probe vectors. However, as mentioned earlier, since we do not use real or simulated data, we are not able to compute the FAR and FRR as in the work of Peeters et al. [56] and thus we cannot base threshold t upon these rates. However, from field practices by GenKey [1], setting t to about $\frac{2}{3}$ of $\mathsf{max}(\mathbb{S})$, has been acknowledged as a reasonable threshold considering measures of the FAR and FRR. Hence, in this work, for each feature set, we fix $t = \frac{2}{3}\mathsf{max}(\mathbb{S})$.

Based on the work of Peeters et al. [56], we use the following feature sets for the runtime assessment of the partially malicious key release protocol:

		k	b	Δ	max(𝕊)	α
fs1	**a**	21	3	0.5	126	84
	b		3	1		42
	c		3	2		21
	d		4	0.5		84
	e		4	1		42
	f		4	2		21

(a) fs1

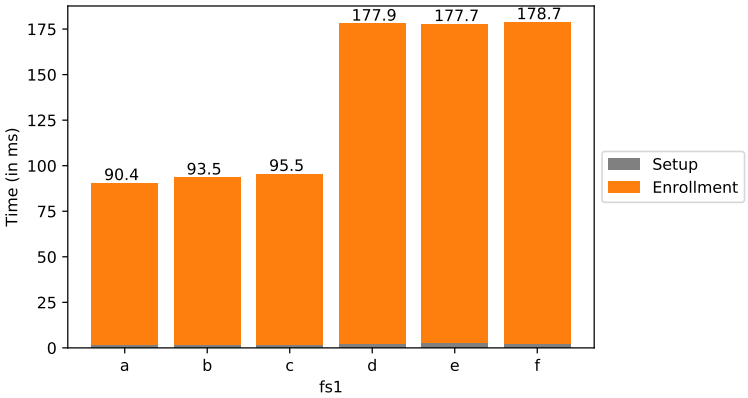
		k	b	Δ	max(𝕊)	α
fs2	**a**	20	3	0.5	120	80
	b		3	1		40
	c		3	2		20
	d		4	0.5		80
	e		4	1		40
	f		4	2		20

(b) fs2

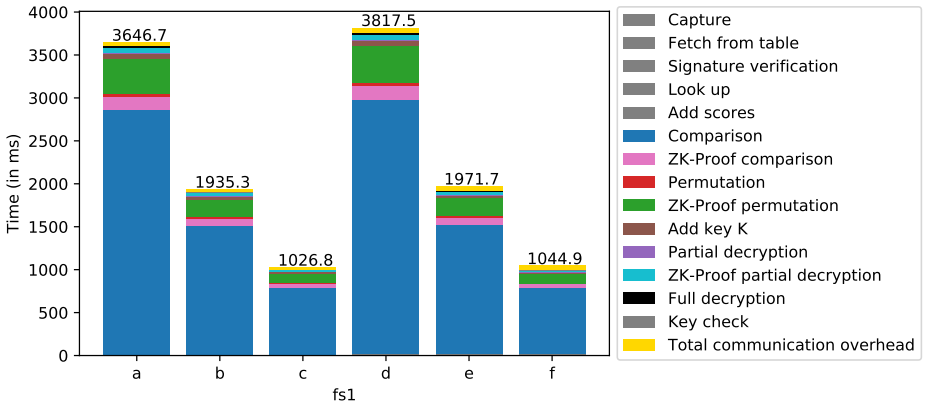
		k	b	Δ	max(𝕊)	α
fs3	**a**	12	3	0.5	72	48
	b		3	1		24
	c		3	2		12
	d		4	0.5		48
	e		4	1		24
	f		4	2		12

(c) fs3

Table 8.3: Feature sets describing size of template $(2^b, k)$, score quantization step size Δ, upper bound of total score domain ($\max(\mathbb{S})$) and number of comparisons (α) for different sub feature sets (a, b, c, e, d, f)

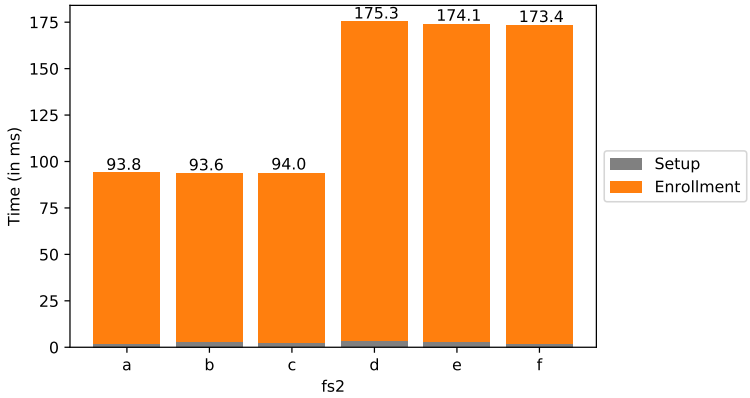


(a) Enrollment Procedure

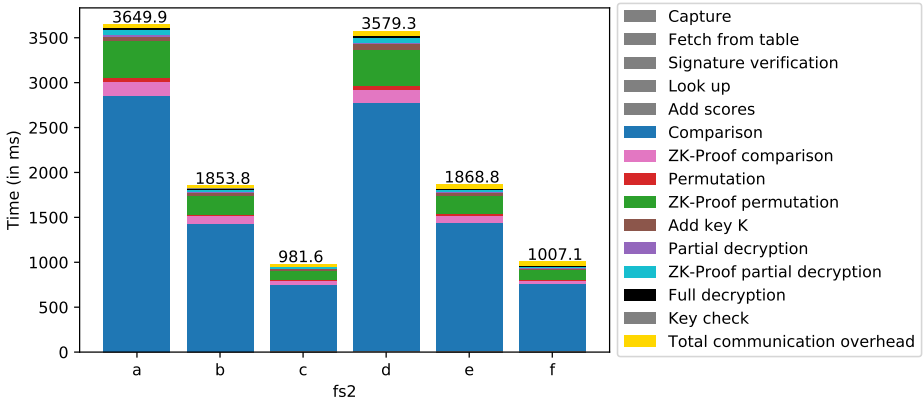


(b) Verification Procedure

Figure 8.1: Average runtime over 10 single executions of the partially malicious key release protocol for feature set fs1

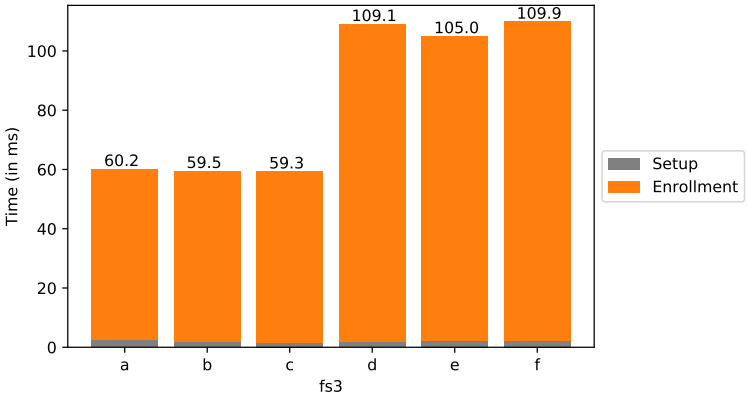


(a) Enrollment Procedure

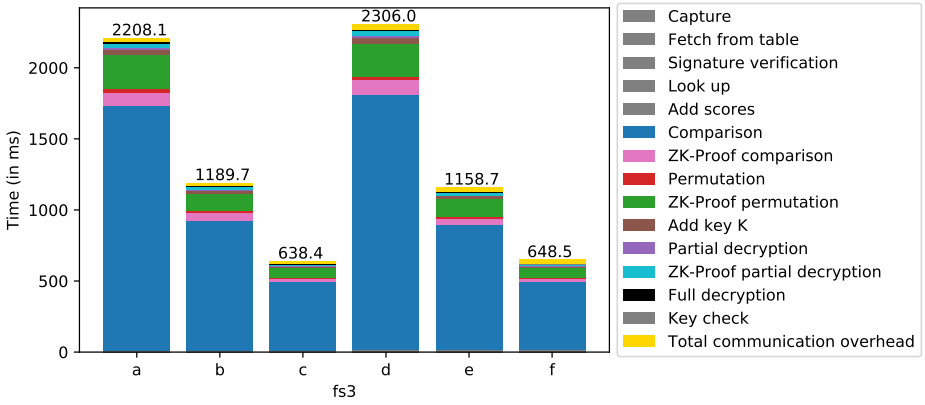


(b) Verification Procedure

Figure 8.2: Average runtime over 10 single executions of the partially malicious key release protocol for feature set fs2



(a) Enrollment Procedure



(b) Verification Procedure

Figure 8.3: Average runtime over 10 single executions of the partially malicious key release protocol for feature set fs3

Figures 8.1, 8.2 and 8.3 show the average run times of 10 protocol runs for each subset a, b, c, d, e and f of the feature sets fs1, fs2 and fs3. The components that have an average runtime of 1ms or less are labeled as grey, to denote their insignificant effect on the total runtime. We refer to Appendix F for the concrete runtimes of the individual components of the partially malicious key release protocol for each feature set.

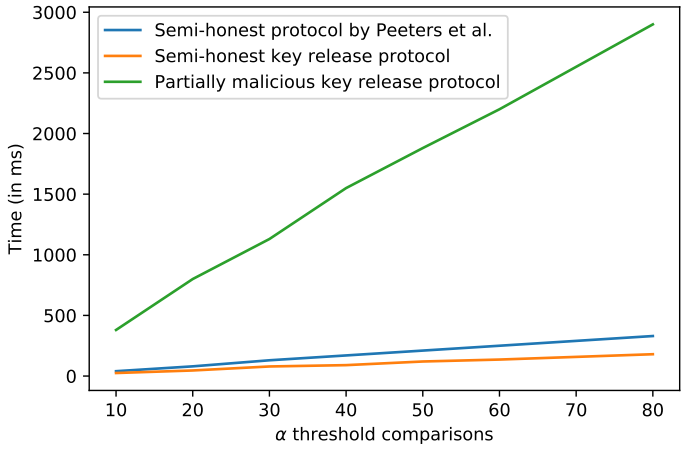


Figure 8.4: Performance of α threshold comparisons

As already mentioned before and as can be seen in Figures 8.1, 8.2 and 8.3, the comparison

operation and thus also α dominates the total runtime of the partially malicious key release protocol. Figure 8.4 shows how the total runtime of α is related to the number of comparisons performed in the protocol, compared to the semi-honest protocol by Peeters et al. [56] and the semi-honest key release protocol. In the semi-honest protocol of Peeters et al. [56], about 250 threshold comparisons per second can be done. For the partially malicious key release protocol, we can do around 25 comparisons per second and for the semi-honest key release protocol about 443 comparisons per second.

8.5 Discussion of Results

Regarding the runtime of the partially malicious key release protocol, first of all observe from Figures 8.1b, 8.2b and 8.3b that the total runtime is dominated by the number of comparisons, as already mentioned before and similar to the work of Peeters et al. [56]. The comparison operation accommodates for about 75.0-78.6% of the total verification runtime. We attribute the high runtime for the comparison to the number of invocations of the Augmented Coin Tossing protocol. As can be observed from Equation 8.10, the ACT protocol is an expensive operation compared to the other building blocks. If we zoom in further on Figures 8.1b, 8.2b and 8.3b, we can see that also the zero-knowledge proof of the permutation makes up a significant part of the verification runtime, namely about 11% of the total verification runtime. The zero-knowledge proof of the comparison accounts for about 4.1-4.4%. The other components in total account for about 6.1-10.7% of the total runtime.

From Figures 8.1a, 8.2a and 8.3a, we can clearly see that for sub feature sets d, e and f, the total enrollment runtime is significantly higher than for a, b and c. Observing Table 8.3, we can conclude that this obviously has to do with the choice of parameter b, which determines the size of the template and thus the number of partially similarity scores that need to be encrypted.

If we consider the individual sub feature sets in the verification phase, we can conclude that c and f are the fastest for each feature set in terms of verification runtime. For c, the average runtime is between 638ms and 1027ms and for f the average runtime is between 649 ms and 1045 ms, depending on the feature sets. This is easily explained by the differences in the values for step size Δ of the score quantization. The greater the step size, the fewer comparisons are needed, which can also be derived from Table 8.3. Similarly, sub feature sets a and d are the slowest, namely their average run times are between 2208 ms and 3650 ms and between 2306 ms and 3818 ms, respectively. However, in this chapter we only consider the efficiency of the partially malicious key release, but not its accuracy. In a realistic setting, efficiency is usually a trade-off with accuracy. In the work of Peeters et al. [56], quantization parameters b and Δ have an effect on the accuracy of the system, in the sense that incrementing b and decrementing Δ yields a higher accuracy.

Moreover, if we compare the performance of the partially malicious key release protocol to the semi-honest key release protocol and the semi-honest protocol by Peeters et al. [56] in the context of the number of α threshold comparisons, we observe that the partially malicious key release protocol is slower than the other two protocols. Moreover, its speed accelerates as the number of threshold comparisons increases compared to the speed of the semi-honest key release protocol and the semi-honest protocol by Peeters et al. [56]. This can be explained by the zero-knowledge protocols and signature verifications that are included in the partially malicious key release protocol.

Finally, if we compare the semi-honest key release protocol to the semi-honest protocol by Peeters et al. [56] in terms of the number of threshold comparisons, we can conclude from Figure 8.4 that the former is faster in terms of verification runtime. Since we have added a key release in the semi-honest key release protocol, the runtime of the protocol should potentially increase (although maybe not significantly). We attribute the speed difference to the differences in the implementation and elliptic curve used.

In a realistic setting, the total verification runtime should be 1s or less in order to satisfy customer usability. As mentioned earlier, in the semi-honest key release protocol and the semi-honest protocol by Peeters et al. [56] we can do about 443 and 250 threshold comparisons per second, respectively. Regarding the feature sets, we can conclude that both protocols are eligible and still not compromise on accuracy. In the partially malicious key release protocol, however, since we can do about 25 threshold comparisons per seconds, only fs1c, fs1f, fs2c, fs2f, fs3b, fs3c, fs3e and fs3f are eligible. Nonetheless, these sub feature sets yield a lower accuracy in realistic settings than the other sub feature sets, so a trade-off between efficiency and accuracy needs to be made.

Chapter 9

Conclusion

In this work, we presented three protocols for biometric verification, i.e. (1) the semi-honest key release protocol, (2) the partially malicious key release protocol and (3) the semi-honest one-round protocol.

The semi-honest key release protocol is an improvement of the semi-honest protocol by Peeters et al. [56] in the sense that it releases a key upon successful comparison. The key is chosen independently from the biometric template and homomorphically binded to the comparison result. Using commitment schemes and zero-knowledge proofs, we have enhanced the semi-honest key release protocol in such a way that it is secure against a sensor device which has been compromised by a semi-honest attacker and a server which has been compromised by a malicious attacker. The semi-honest one-round protocol is based on the BV-scheme, which supports both homomorphic additions and multiplications to enable complete comparison on the server side. The semi-honest one-round protocol serves as a stepping stone for protocols where the sensor device is more likely to be compromised than the server, since we have only provided a theoretic overview.

In both the semi-honest and partially malicious key release protocol, we found that the bottleneck in terms of efficiency is the comparison operation, which accommodates for about 75.0-78.6% of the total runtime of the partially malicious key release protocol. In the semi-honest key release protocol, we can do about 443 comparisons per second, compared to about 250 comparisons per second in the semi-honest protocol of Peeters et al. [56]. In the partially malicious key release protocol, we can only do about 25 comparisons per second. However, since efficiency is usually a trade-off with accuracy, the partially malicious key release protocol can be practical (< 1s) if we choose certain sets of parameters for which the accuracy is low.

We conclude that the semi-honest key release protocol is very efficient in realistic settings, but the partially malicious key release protocol is only practical if the accuracy is low.

9.1 Limitations and Future Work

Regarding the semi-honest one-round protocol, in the current setting, it is not possible to transform the protocol into a (partially) malicious one in a similar way as the semi-honest key release protocol, since the BV-scheme does not provide support for zero-knowledge proofs and commitment schemes. A literature study is needed to search for an approach on how to make somewhat homomorphic encryption scheme based protocols secure against malicious attackers.

A limitation of the semi-honest and partially malicious key release protocol is the experimental analysis. In the experimental analysis, we have used randomly generated data, rather than simulated or real data. That way, feature quality parameter ρ in the work of Peeters et al. [56] does not play a role in the partially malicious key release protocol. The performance of both protocols could be assessed as a future work using the simulated data of Peeters et al. [56], to obtain a more accurate comparison between the three protocols.

Other improvements of this work regard several performance optimizations on the semi-honest and partially malicious key release protocol. First of all, the comparison part dominates the total runtime of both protocols. The protocols can gain efficiency in the comparison of the total score with the threshold, by using a ciphertext comparison method, such as the work of Peng et al. [14]. Secondly, in the current setting, we use the K-233 elliptic curve, which is a standard Weierstrass elliptic curve as chosen by NIST [41]. Other curves, such the New Weierstrass curve or the Twisted Edwards curve could potentially yield a significant speed up for both protocols. The former curve is 1.4 times and

the latter is 1.9 times faster than the fastest NIST curve on a 128-bit security level [12].

The most interesting enhancement of this work, however, is the realization of a full malicious key release protocol. In this section, we will give some pointers for transforming the partially malicious key release protocol into a full malicious one.

First of all, in order to transform the partially malicious key release protocol into a full malicious key release protocol, the sensor also needs to prove correct execution of his steps. Prior to the enrollment and verification procedure, the sensor and server need to engage in a key setup. We propose this key setup is done in the same way as in the rest of this work. Hence, there will be no proofs in this part, since we can check later in the protocol if the correct shared public key has been used for encryption. In the enrollment and verification procedure, some of the steps of the sensor device can be proven in a similar way as some of the steps of the server. For instance, Step 3 can be performed by the Augmented Coin Tossing protocol and Step 17 can be performed by enabling the server to compute $[[S]]$ itself simultaneously, similar to Steps 19 and 26. Please note that malicious behavior in the biometric domain, such as fingerprint spoofing in Step 1 and 10, is outside the scope of this research, therefore we will not prove correctness of these steps.

Secondly, recall that the used zero-knowledge protocols in Steps 22, 25 and 30 of the partially malicious key release protocol are honest-verifier zero-knowledge. For the full malicious key release protocol, we need zero-knowledge protocols which are secure against either a malicious prover or a malicious verifier. We propose that the zero-knowledge protocols used in this work are transformed into non-interactive zero-knowledge proofs, as obtained by applying the Fiat-Shamir heuristic (see Section 2.3.2.2). Recall that the strong variant of the Fiat-Shamir heuristic includes the proof statement in the hash function [8]. Transforming the zero-knowledge protocols in this work using the strong Fiat-Shamir heuristic can be done using standard methods of libscapi [2]. Alternative maliciously secure zero-knowledge proofs can be found in the direction of zk-STARKs or Bulletproofs, however, more literature study needs to be undertaken in this domain. The signatures in the partially malicious key release protocol are instantiated as Schnorr signatures, which are already secure in the random oracle model [10, 62]. Hence, Steps 6, 7, 14 and 15 do not have to be changed.

Finally, what remains is to find a maliciously secure method to prove correct ElGamal encryption of K under pk_{shared} in Step 4, correct AES encryption of 1 under key K in Step 5, a correct lookup procedure in Step 16 and a correct decryption of $[\vec{B}]$ under key sk_{ss} in Step 31. Step 31 cannot be performed using the equality of exponents proof, since it requires the server to know $\vec{B}$, which obviously leaks the outcome of the protocol. For Step 4 and Step 31, we propose that an interested reader looks into the topics of verifiable encryption and decryption.

Bibliography

[1] Genkey. `http://genkey.com`. [Online; accessed 07-December-2019].

[2] libscapi. `https://biulibscapi.readthedocs.io/en/latest/intro.html`, 2017. [Online; accessed 25-October-2019].

[3] M. Abdalla, J. H. An, M. Bellare, and C. Namprempre. From identification to signatures via the fiat-shamir transform: Minimizing assumptions for security and forward-security. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 418–433. Springer, 2002.

[4] D. F. Aranha, A. Faz-Hernández, J. López, and F. Rodríguez-Henríquez. Faster implementation of scalar multiplication on koblitz curves. In A. Hevia and G. Neven, editors, *Progress in Cryptology – LATINCRYPT 2012*, pages 177–193, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[5] E. Barker, W. Barker, W. Burr, W. Polk, and M. Smid. Nist special publication 800-57. *NIST Special publication*, 800(57):1–142, 2007.

[6] S. Bartolini, I. Branovi, R. Giorgi, and E. Martinelli. Effects of instruction-set extensions on an embedded processor: A case study on elliptic curve cryptography over gf(2m). *Computers, IEEE Transactions on*, 57:672 – 685, 06 2008.

[7] F. Benhamouda, S. Krenn, V. Lyubashevsky, and K. Pietrzak. Efficient zero-knowledge proofs for commitments from learning with errors over rings. In *European Symposium on Research in Computer Security*, pages 305–325. Springer, 2015.

[8] D. Bernhard, O. Pereira, and B. Warinschi. How not to prove yourself: Pitfalls of the fiat-shamir heuristic and applications to helios. In X. Wang and K. Sako, editors, *Advances in Cryptology – ASIACRYPT 2012*, pages 626–643, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[9] D. Boneh. *Digital Signature Standard*, pages 347–347. Springer US, Boston, MA, 2011.

[10] D. Boneh. *Schnorr Digital Signature Scheme*, pages 1082–1083. Springer US, Boston, MA, 2011.

[11] D. Boneh, E.-J. Goh, and K. Nissim. Evaluating 2-dnf formulas on ciphertexts. In J. Kilian, editor, *Theory of Cryptography*, pages 325–341, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[12] J. W. Bos, C. Costello, P. Longa, and M. Naehrig. Selecting elliptic curves for cryptography: an efficiency and security analysis. *Journal of Cryptographic Engineering*, 6(4):259–286, Nov 2016.

[13] C. Bosch, A. Peter, P. Hartel, and W. Jonker. Sofir: Securely outsourced forensic image recognition. pages 2694–2698, 05 2014.

[14] C. Boyd, E. Dawson, B. Lee, and K. Peng. Ciphertext comparison, a new solution to the millionaire problem. 12 2005.

[15] Z. Brakerski and V. Vaikuntanathan. Fully homomorphic encryption from ring-lwe and security for key dependent messages. In *Advances in Cryptology – CRYPTO 2011*, pages 505–524, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[16] J. Bringer, O. El Omri, C. Morel, and H. Chabanne. Boosting gshade capabilities: New applications and security in malicious setting. In *Proceedings of the 21st ACM on Symposium on Access Control Models and Technologies*, SACMAT '16, pages 203–214, New York, NY, USA, 2016. ACM.

[17] J. Camenisch and M. Stadler. Efficient group signature schemes for large groups. In B. S. Kaliski, editor, *Advances in Cryptology — CRYPTO '97*, pages 410–424, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

[18] D. Chaum and T. P. Pedersen. Wallet databases with observers. In E. F. Brickell, editor, *Advances in Cryptology — CRYPTO' 92*, pages 89–105, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.

[19] I. Damgård. *Commitment Schemes and Zero-Knowledge Protocols*, pages 63–86. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999.

[20] I. Damgård. On sigma-protocols. *Lecture Notes, University of Aarhus, Department for Computer Science*, 2002.

[21] E. De Win, S. Mister, B. Preneel, and M. Wiener. On the performance of signature schemes based on elliptic curves. In *International Algorithmic Number Theory Symposium*, pages 252–266. Springer, 1998.

[22] G. Di Crescenzo, J. Katz, R. Ostrovsky, and A. Smith. Efficient and non-interactive non-malleable commitment. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 40–59. Springer, 2001.

[23] Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation). *Official Journal of the European Union*, L119:1–88, May 2016.

[24] J. Furukawa. Efficient and verifiable shuffling and shuffle-decryption. *IEICE Transactions*, 88-A:172–188, 01 2005.

[25] J. Furukawa, H. Miyauchi, K. Mori, S. Obana, and K. Sako. An implementation of a universally verifiable electronic voting scheme based on shuffling. In M. Blaze, editor, *Financial Cryptography*, pages 16–30, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[26] J. Furukawa and K. Sako. An efficient scheme for proving a shuffle. In J. Kilian, editor, *Advances in Cryptology — CRYPTO 2001*, pages 368–387, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[27] C. Gentry, S. Halevi, and V. Vaikuntanathan. A simple bgn-type cryptosystem from lwe. In H. Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, pages 506–522, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[28] O. Goldreich. *Foundations of Cryptography - Basic Applications*, volume 2. Cambridge University Press, 2004.

[29] O. Goldreich. *Foundations of Cryptography - Basic Tools*, volume 1. Cambridge University Press, 2007.

[30] O. Goldreich, S. Micali, and A. Wigderson. A completeness theorem for protocols with honest majority. *Conference Proceedings of the Annual ACM Symposium on Theory of Computing*, 01 1987.

[31] M. Gomez-Barrero, E. Maiorana, J. Galbally, P. Campisi, and J. Fierrez. Multi-biometric template protection based on homomorphic encryption. *Pattern Recognition*, 67, 01 2017.

[32] M. Gomez-Barrero, C. Rathgeb, G. Li, R. Ramachandra, J. Galbally, and C. Busch. Multi-biometric template protection based on bloom filters. *Information Fusion*, 42:37 – 50, 2018.

[33] J. Groth. A verifiable secret shuffle of homomorphic encryptions. *Journal of Cryptology*, 23(4):546–579, Oct 2010.

[34] J. Groth and Y. Ishai. Sub-linear zero-knowledge argument for correctness of a shuffle. In N. Smart, editor, *Advances in Cryptology – EUROCRYPT 2008*, pages 379–396, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[35] S. Gueron, S. Johnson, and J. Walker. Sha-512/256. In *2011 Eighth International Conference on Information Technology: New Generations*, pages 354–358. IEEE, 2011.

[36] C. Hazay and Y. Lindell. *Efficient Secure Two-Party Protocols: Techniques and Constructions*, pages 147–175. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[37] ISO/IEC JTC1 SC27 Security Techniques. ISO/IEC 24745:2011. *Biometric Information Protection*, 2011.

[38] A. Jain, K. Nandakumar, and A. Nagar. Biometric template security. *EURASIP J. Adv. Signal Process*, 2008:113:1–113:17, Jan. 2008.

[39] A. Jain, A. Ross, and K. Nandakumar. *Introduction to Biometrics*. SpringerLink : Bücher. Springer US, 2011.

[40] M. Just. *Schnorr Identification Protocol*, pages 1083–1083. Springer US, Boston, MA, 2011.

[41] C. F. Kerry and P. D. Gallagher. Fips pub 186-4 federal information processing standards publication digital signature standard (dss), Jul 2013.

[42] T. Lange. Koblitz curve cryptosystems. *Finite Fields and Their Applications*, 11(2):200 – 229, 2005.

[43] L. Li, A. A. A. El-Latif, and X. Niu. Elliptic curve elgamal based homomorphic image encryption scheme for sharing secret images. *Signal Processing*, 92(4):1069 – 1078, 2012.

[44] T. Lim. A study of koblitz curves. 2015.

[45] Y. Lindell. *How to Simulate It – A Tutorial on the Simulation Proof Technique*, pages 277–346. Springer International Publishing, Cham, 2017.

[46] Y. Lindell and B. Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. volume 28, pages 52–78, 06 2007.

[47] J. Malik, D. Girdhar, R. Dahiya, and G. Sainarayanan. Reference threshold calculation for biometric authentication. *IJ Image, Graphics and Signal Processing*, 2:46–53, 2014.

[48] E. Martiri, M. Gomez-Barrero, B. Yang, and C. Busch. Biometric template protection based on bloom filters and honey templates. *IET Biometrics*, 6(1):19–26, 2017.

[49] K. Nandakumar and A. K. Jain. Biometric template protection: Bridging the performance gap between theory and practice. *IEEE Signal Processing Magazine*, 32(5):88–100, 2015.

[50] S. Nazari, M.-S. Moin, and H. R. Kanan. Securing templates in a face recognition system using error-correcting output code and chaos theory. *Computers & Electrical Engineering*, 72:644 – 659, 2018.

[51] G. Neven, N. Smart, and B. Warinschi. Hash function requirements for schnorr signatures. *Journal of Mathematical Cryptology*, 3, 05 2009.

[52] M. Ogburn, C. Turner, and P. Dahal. Homomorphic encryption. *Procedia Computer Science*, 20:502 – 509, 2013. Complex Adaptive Systems.

[53] K. Ohta and T. Okamoto. On concrete security treatment of signatures derived from identification. In *Annual International Cryptology Conference*, pages 354–369. Springer, 1998.

[54] V. M. Patel, N. K. Ratha, and R. Chellappa. Cancelable biometrics: A review. *IEEE Signal Processing Magazine*, 32(5):54–65, Sep. 2015.

[55] T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In J. Feigenbaum, editor, *Advances in Cryptology — CRYPTO '91*, pages 129–140, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.

[56] J. Peeters, A. Peter, and R. N. J. Veldhuis. Fast and accurate likelihood ratio based biometric comparison in the encrypted domain. May 2017.

[57] C. Rathgeb, F. Breitinger, C. Busch, and H. Baier. On application of bloom filters to iris biometrics. *IET Biometrics*, 3(4):207–218, 2014.

[58] D. Sadhya and S. K. Singh. Providing robust security measures to bloom filter based biometric template protection schemes. *Computers & Security*, 67:59 – 72, 2017.

[59] M. Saikia. Implementation of elgamal elliptic curve cryptography over prime field using c. 02 2014.

[60] N. Schmid, M. V. Ketkar, H. Singh, and B. Cukic. Performance analysis of iris based identification system at the matching, 01 2006.

[61] B. Schoenmakers. Lecture notes cryptographic protocols, 2019.

[62] N. Smart. *Cryptography Made Simple*. Information Security and Cryptography. Springer, 2016.

[63] D. Stritzl. Privacy-preserving matching using bloom filters: an analysis and an encrypted variant. Master's thesis, University of Twente, 2019.

[64] U. Uludag, S. Pankanti, S. Prabhakar, and A. K. Jain. Biometric cryptosystems: issues and challenges. *Proceedings of the IEEE*, 92(6):948–960, June 2004.

[65] P. C. Van Oorschot, A. J. Menezes, and S. A. Vanstone. *Handbook of applied cryptography*. Crc Press, 1996.

[66] C. Zhang, L. Zhu, and C. Xu. Ptbi: An efficient privacy-preserving biometric identification based on perturbed term in the cloud. *Information Sciences*, 409-410:56 – 67, 2017.

[67] C. Zhao, S. Zhao, M. Zhao, Z. Chen, C.-Z. Gao, H. Li, and Y. Tan. Secure multi-party computation: Theory, practice and applications. *Information Sciences*, 476:357 – 372, 2019.

[68] ZKProof. Zkproof community reference, December 2019.

Appendix A

Correctness Proof of Semi-Honest Key Release Protocol

In this section, we will prove that the protocol as depicted in Figure 5.2 correctly computes the following functionality:

$$f(\vec{p}, \perp) = (K \cdot \Big(\Sigma(\mathsf{LookUp}(\vec{p}, [[\mathbf{T}_u]])) \geq t\Big), \perp) \tag{A.1}$$

The sensor device has private input $x = \vec{p}$. Note that u is not private input of the sensor, since u is sent to the server in the clear and is thus considered public knowledge. The server has no private input, hence we denote its input by $y = \perp$. Recall that although the lookup tables that $\mathbf{T}_u$ is comprised of are public knowledge, $\mathbf{T}_u$ is encrypted, in order to obfuscate the rows that were selected from the lookup tables [56]. Hence, $\mathbf{T}_u$ is no private input of the server, since the server does not own the secret key to decrypt $[[\mathbf{T}_u]]$. Also, $[[\mathbf{T}_u]]$ is encrypted and thus the encrypted form is considered public knowledge. The sensor obtains a key based on whether or not the biometric identifier presented to it was successfully verified. The server has no output.

In Section A.1 we will provide a proof for the correctness of $f_1(\vec{p}, \perp) = K \cdot \Big(\Sigma(\mathsf{LookUp}(\vec{p}, [[\mathbf{T}_u]])) \geq t\Big)$, in other words, the output of functionality f for the sensor device. Then in Section A.2, we will give a correctness proof for $f_2(\vec{p}, \perp) = \perp$, in other words, the output of functionality f for the verification server. For the correctness proofs, we will concatenate the enrollment procedure and the verification procedure (see Figures 5.1 and 5.2) as one protocol computing f.

A.1 Sensor Device

We will prove that the protocol as depicted in Figures 5.1 and 5.2 correctly computes function $f_1(\vec{p}, \perp) = K \cdot \Big(\Sigma(\mathsf{LookUp}(\vec{p}, [[\mathbf{T}_u]])) \geq t\Big)$. As mentioned before, we will concatenate the enrollment and verification procedure as one protocol computing f_2.

Proof. In case the sensor device is Party 1 and the verification server Party 2, let functionality $g_{x,y}(\vec{p}, \perp)$ be the intermediate functionality for Party x at Step y resembling the protocol in Figures 5.1 and 5.2. We will prove that at the end of a valid protocol run, $g_{1,22}(\vec{p}, \perp) = f_1(\vec{p}, \perp)$ applies.

Step 1-11: Since Steps 1-11 have no influence on the correctness of functionality $g_{x,y}(\vec{p}, \perp)$, we assume these steps are carried out correctly in the semi-honest model. More specifically, we assume that the enrollment party and sensor correctly capture a biometric identifier and that the enrollment party correctly constructs a valid template using ElGamal, which keeps the elements intact, similar to the work of Peeters et al. [56]. Also, we assume the sensor correctly chooses k and computes $\mathsf{AES}_K(1)$. On the server side, we assume that $(u, [[\mathbf{T}_u]], [[k]], \mathsf{AES}_K(1))$ are correctly stored in the database and correctly fetched from the database upon later retrieval.

Step 12: At this point, we can start defining functionality g, since we have obtained the private inputs for both parties:

$$g_{1,12}(\vec{p}, \perp) = [[\vec{s}]] = \{\mathsf{LookUp}(\vec{p}, [[\mathbf{T}_u]])\} \tag{A.2}$$

However, since the result of the look up procedure yields a vector with encrypted similarity scores, we can write:

$$g_{1,12}(\vec{p}, \perp) = [[\{\mathsf{LookUp}(\vec{p}, \mathbf{T}_u)\}]] \tag{A.3}$$

Step 13: Since $[[S]]$ is the sum of all $[[s_i]]$ and thus $[[S]] = [[\Sigma(\mathsf{LookUp}(\vec{p}, \mathbf{T}_u))]]$, we can write:

$$g_{1,13}(\vec{p}, \perp) = [[\Sigma(\mathsf{LookUp}(\vec{p}, \mathbf{T}_u))]] \tag{A.4}$$

Step 14: $[[S]]$ is sent to the server and thus we can write:

$$g_{2,14}(\vec{p}, \perp) = [[\Sigma(\mathsf{LookUp}(\vec{p}, \mathbf{T}_u))]] \tag{A.5}$$

Step 15: The compare function checks if $[[S \geq t]]$ by computing $[[S-t-i]]$ for each $0 \leq i \leq \mathsf{max}(\mathbb{S})-t$ and additionally multiplies the corresponding values with blinding factor r_i. Therefore we can write

$$g_{2,15}(\vec{p}, \perp) = [[r_i(\Sigma(\mathsf{LookUp}(\vec{p}, \mathbf{T}_u)) - t - i)]] \mid \forall 0 \leq i \leq \mathsf{max}(\mathbb{S}) - t \tag{A.6}$$

See the work of Peeters et al. [56] for the correctness proof of this step.

Step 16: Permutation function π is applied, so we can write:

$$g_{2,16}(\vec{p}, \perp) = \pi([[r_i(\Sigma(\mathsf{LookUp}(\vec{p}, \mathbf{T}_u)) - t - i)]]) \mid \forall 0 \leq i \leq \mathsf{max}(\mathbb{S}) - t \tag{A.7}$$

Step 17: $[[k]]$ and $[[\mathsf{AES}_K(1)]]$ are fetched from the database, which has no influence on function g, so therefore we disregard this step.

Step 18: $[[k]]$ is multiplied with the encrypted result set $[[\vec{C}]]$, which is the outcome of function g in Step 11, and thus we can write:

$$g_{2,18}(\vec{p}, \perp) = \pi([[r_i(\Sigma(\mathsf{LookUp}(\vec{p}, \mathbf{T}_u)) - t - i)]]) \cdot [[k]] \mid \forall 0 \leq i \leq \mathsf{max}(\mathbb{S}) - t \tag{A.8}$$

Step 19: Decryption function D_1 is performed by the server, so we can write

$$g_{2,19}(\vec{p}, \perp) = \pi([r_i(\Sigma(\mathsf{LookUp}(\vec{p}, \mathbf{T}_u)) - t - i)]) \cdot [k] \mid \forall 0 \leq i \leq \mathsf{max}(\mathbb{S}) - t \tag{A.9}$$

Step 20: $[\vec{B}]$ and $\mathsf{AES}_K(1)$ are sent to the sensor device, but g does not change, and thus the intermediate function becomes $g_{1,21} = g_{2,20}$.

Step 21: Decryption function D_2 is performed by the sensor, so we write

$$g_{1,21}(\vec{p}, \perp) = K \cdot \pi(r_i(\Sigma(\mathsf{LookUp}(\vec{p}, \mathbf{T}_u)) - t - i)) \mid \forall 0 \leq i \leq \mathsf{max}(\mathbb{S}) - t \tag{A.10}$$

Step 22: In the protocol, the sensor checks statement $\Big(\exists B_i : \mathsf{AES\text{-}D}_{B_i}(\mathsf{AES}_K(1)) = 1\Big)$ and in case it is true, it extracts the corresponding B_i as key K. Moreover, we can remove permutation function π, because it does not influence the result of the key extraction. Therefore, we can write

$$g_{1,22}(\vec{p}, \perp) = K \cdot \Big(\exists B_i : \mathsf{AES\text{-}D}_{B_i}(\mathsf{AES}_K(1)) = 1\Big) \tag{A.11}$$

where $B_i = K \cdot r_i(\Sigma(\mathsf{LookUp}(\vec{p}, \mathbf{T}_u)) - t - i)$ for $0 \leq i \leq \mathsf{max}(\mathbb{S}) - t$. Note that $B_i = K$ if $\Big(\exists B_i : \mathsf{AES\text{-}D}_{B_i}(\mathsf{AES}_K(1)) = 1\Big)$, since the terms K and $\Sigma(r_i(\mathsf{LookUp}(\vec{p}, \mathbf{T}_u)) - t - i)$ are encoded as powers of generator g.

Now we can finally prove that $f_1(\vec{p}, \perp) = g_{1,22}(\vec{p}, \perp)$. We first need to make a case distinction:

1. $f_1(\vec{p}, \perp) = K$
2. $f_1(\vec{p}, \perp) = 0$

For case 1, we can construct the following subproof:

Subproof. In case $f_1(\vec{p}, \perp) = K$, the term $\Sigma(\mathsf{LookUp}(\vec{p}, [[\mathbf{T}_u]])) \geq t$ should be true. Let $X_i = r_i(\Sigma(\mathsf{LookUp}(\vec{p}, \mathbf{T}_u)) - t - i)$ for $0 \leq i \leq \mathsf{max}(\mathbb{S}) - t$. We can now state $\exists X_i : X_i = 0$. Recall that X_i is encoded as g^{X_i}, so in case $X_i = 0$, $g^{X_i} = 1$ and then $B_i = K$. In this case, $\exists B_i : B_i = K$ and thus the term $\left(\exists B_i : \mathsf{AES\text{-}D}_{B_i}(\mathsf{AES}_K(1)) = 1\right)$ is true. Therefore we can conclude that $g_{1,23}(\vec{p}, \perp) = K = f_1(\vec{p}, \perp)$. □

For case 2, we can construct the following subproof:

Subproof. In case $f_1(\vec{p}, \perp) = 0$, the term $\Sigma(\mathsf{LookUp}(\vec{p}, [[\mathbf{T}_u]])) \geq t$ should be false. Let X_i be as denoted previously. We can now state $\nexists X_i : X_i = 0$. In this case, $\nexists B_i : B_i = K$ and thus the term $\left(\exists B_i : \mathsf{AES\text{-}D}_{B_i}(\mathsf{AES}_K(1)) = 1\right)$ is false. Therefore we can conclude that $g_{1,23}(\vec{p}, \perp) = 0 = f_1(\vec{p}, \perp)$. □

■

A.2 Server

The proof that the protocol as depicted in Figure 5.2 correctly computes function $f_2(\vec{p}, \perp) = \perp$ is trivial, as the protocol does not have any output for the server and thus we can conclude that $f_2(\vec{p}, \perp) = \perp$.

Appendix B

Security Proof of Semi-Honest Key Release Protocol

Recall from Definition 3 that a deterministic function f securely computes protocol π in the semi-honest model if there exist probabilistic polynomial-time algorithms, S_1 and S_2, such that:

$$\{S_1(x, f_1(x, y))\}_{x,y\in\{0,1\}^*} \stackrel{c}{\equiv} \{\mathsf{VIEW}_1^{\pi}(x, y)\}_{x,y\in\{0,1\}^*} \tag{2.3}$$

$$\{S_2(y, f_2(x, y))\}_{x,y\in\{0,1\}^*} \stackrel{c}{\equiv} \{\mathsf{VIEW}_2^{\pi}(x, y)\}_{x,y\in\{0,1\}^*} \tag{2.4}$$

where $|x| = |y|$ and $f_1(x, y)$ and $f_2(x, y)$ are the outputs of Party 1 and 2, respectively.

Recall from Equation A.1 that functionality f for inputs $x = \vec{p}$ and $y = \perp$ is defined as follows:

$$f(\vec{p}, \perp) = (K \cdot \Big(\Sigma(\mathsf{LookUp}(\vec{p}, [[\mathbf{T}_u]])) \geq t\Big), \perp) \tag{A.1}$$

In order to obtain a full proof on the security of the key release protocol from Section 5.2 in the semi-honest model as defined in Definition 3, we need to distinguish between two cases and prove their security separately:

(a) The server is honest and the sensor device is compromised by a semi-honest attacker

(b) The sensor device is honest and the server is compromised by a semi-honest attacker

For case (a), we proof Equation 2.3 to show that the protocol is secure against a semi-honest sensor device, where we assume the sensor device has been compromised by a semi-honest attacker and the server is honest. Subsequently, for case (b) we perform a similar proof for Equation 2.4 where we assume the server is compromised by a semi-honest attacker and the sensor is honest. For the security proofs, we will concatenate the enrollment procedure and the verification procedure as one protocol computing f.

B.1 Case (a)

Proof. Assume that the sensor device is Party 1 and let $f_1'(\vec{p}, \perp)$ be the functionality that is computed by S_1. Similar to Peeters et al. [56], we assume that the sensor device is located in an off line setting and the sensor device follows the protocol, so Steps 1-7 were performed in an honest setting.

Observe that if the term $\Sigma(\mathsf{LookUp}(\vec{p}, [[\mathbf{T}_u]])) \geq t$ is true, then $f(\vec{p}, \perp) = K$, else $f(\vec{p}, \perp) = 0$ (false). Hence, for the verification procedure from Step 8 onwards, we need to distinguish between two cases:

1. $f_1(\vec{p}, \perp) = K$
2. $f_1(\vec{p}, \perp) = 0$

For both cases we need to prove that S_1 is computationally indistinguishable from the real model view.

In case $f_1(\vec{p}, \perp) = K$, we can construct the following subproof:

Subproof. For S_1, in Figure 5.2, replace messages u and $[[S]]$ in Steps 9 and 14 with u' and $[[S']]$, respectively, such that $S' = \mathsf{max}(\mathbb{S})$. Now $[[S' \geq t]]$ holds and thus $\exists B'_i : B'_i = K$. We can therefore state that $f'_1(\vec{p}, \perp) = K = f_1(\vec{p}, \perp)$. The sensor device knows $\mathsf{max}(\mathbb{S})$, because it is a public value. Also, both the sensor device and the verification server have encryption keys pk_{shared} and pk_{sv}, so the sensor is able to encrypt S' in polynomial time. Therefore, we can state that S_1 is a polynomial-time algorithm. □

In case $f(\vec{p}, \perp) = 0$, we can construct the following subproof:

Subproof. For S_1, in Figure 5.2, replace messages u and $[[S]]$ in Steps 11 and 14 with u' and $[[S']]$, respectively, such that $S' = \mathsf{min}(\mathbb{S})$. Now $[[S' < t]]$ holds and thus $\nexists B'_i : B'_i = K$. We can therefore state that $f'_1(\vec{p}, \perp) = 0 = f_1(\vec{p}, \perp)$. Similar to the previous case, the sensor device knows $\mathsf{min}(\mathbb{S})$ and is able to encrypt S' and $\vec{B}$ in polynomial time and thus S_1 is a polynomial-time algorithm. □

For both cases, we can state that ElGamal is secure under a chosen plaintext attack [62] and that the sensor does not possess sk_{shared} due to the threshold encryption system, thus S_1 is computationally indistinguishable from the real model view. We can therefore conclude that Equation 2.3 holds and thus a compromised sensor device is not able to learn anything during the protocol run, except from its own input and output value. ■

B.2 Case (b)

Proof. Assume that the server is Party 2 and let S_2 be the algorithm that computes f'_2. In Figure 5.2, replace messages $[[\mathbf{T}_u]], [\vec{B}]$ and $\mathsf{AES}_K(1)$ in Steps 11 and 20 with $[[\mathbf{T}'_u]], [\vec{B}']$ and $\mathsf{AES}_{K'}(1)$, respectively, such that $f'_2(\vec{p}, \perp) = f_2(\vec{p}, \perp) = \perp$. Since the verification server has no output ($\perp$), we can choose $\mathbf{T}'_u, \vec{B}'$ and K' randomly and encrypt these. This can all be done in polynomial time, since the server knows pk_{shared} and pk_{ss}. In all cases, $f'_2(\vec{p}, \perp) = f_2(\vec{p}, \perp) = \perp$ holds. Again, since ElGamal is secure under a chosen plaintext attack [62] and the server does not possess sk_{shared} due to the threshold encryption system, S_2 is computationally indistinguishable from the real model view. We can therefore conclude that Equation 2.4 holds and a compromised verification server is not able to learn anything during the protocol, except from its own input and output value. ■

Appendix C

Requirements Proof of Semi-Honest Key Release Protocol

We will prove the protocol in Figure 5.2 complies to the following requirements:

1. The verification server should not be able to learn anything about key K.
2. K should be consistent for every biometric probe $\vec{p}$ belonging to the same identity u.

C.1 Requirement 1

Proof. In Step 3 of the enrollment procedure (see Figure 5.1), an integer $k \in_R \mathbb{Z}_q$ is chosen which is first encoded as $K \leftarrow g^k$. Subsequently, K is encrypted using ElGamal in Step 4. Since ElGamal is secure under a chosen plaintext attack [62], keys $K_0, ..., K_m$ in the database at the server side will be computationally indistinguishable and thus the verification server will not be able to learn anything about K. ■

C.2 Requirement 2

Theorem C.2.1. *Given two biometric probes $\vec{p}$ and $\vec{p'}$ where $u = u'$. If $S \geq t$ and $S' \geq t$ applies, then $K = K'$.*

Proof. We can distinguish between two cases:

1. $\vec{p} = \vec{p'}$
2. $\vec{p} \neq \vec{p'}$

For both cases, we have the following protocol emulation of Figure 5.2 for $\vec{p'}$:

Sensor Device		Verification Server
(8) $u, \vec{p'} \leftarrow \mathsf{Capture}()$		
	(9) $\xrightarrow{\text{u}}$	
		(10) $[[\mathbf{T}_u]] \leftarrow \mathsf{FetchTemplate}(u)$
	(11) $\xleftarrow{[[\mathbf{T}_u]]}$	
(12) $[[\vec{s'}]] \leftarrow \mathsf{LookUp}(\vec{p'}, [[\mathbf{T}_u]])$		
(13) $[[S']] \leftarrow [[0 + \sum_{i=0}^{k-1} s'_i]]$		
	(14) $\xrightarrow{[[S']]}$	
		(15) $[[\vec{C'}]] \leftarrow \mathsf{Compare}([[S']], t)$
		(16) $\pi'([[\vec{C'}]])$
		(17) $([[k']], \mathsf{AES}_{K'}(1)) \leftarrow \mathsf{FetchKey}(u)$
		(18) $[[\vec{B'}]] \leftarrow [[\vec{C'}]] \cdot [[k']]$
		(19) $[\vec{B'}] \leftarrow \mathsf{D}_1([[\vec{B'}]])$
	(20) $\xleftarrow{([\vec{B'}], \mathsf{AES}_{K'}(1))}$	
(21) $\vec{B'} \leftarrow \mathsf{D}_2([\vec{B'}])$		
(22) $\exists B'_i : \mathsf{AES\text{-}D}_{B'_i}(\mathsf{AES}_{K'}(1)) = 1$ $\implies S' \geq t$		

Figure C.1: Emulation of protocol from Figure 5.2 for $\vec{p'}$

In case $\vec{p} = \vec{p'}$, we can construct the following subproof:

Subproof. Case 1 seems trivial, since $\vec{p} = \vec{p'}$. The only non-triviality is in the permutation function of Step 16. If $\vec{p} = \vec{p'}$, then $[[C'_j]] = [[C_j]]$ in Step 15. Due to the permutation, $[[C'_m]] = [[C_n]]$ holds after Step 16. However, since $u' = u$, it follows that $([[k']], \mathsf{AES}_{K'}(1)) = ([[k]], \mathsf{AES}_K(1))$ in Step 17. Combining the previous statements, we can conclude that $\exists B'_i, B_i : B'_i = B_i$ in Step 18 and thus $K' = K$. □

In case $\vec{p} \neq \vec{p'}$, we can construct the following subproof:

Subproof. In Step 15, the server computes $[[\vec{C'}]]$. Since we assume that $S' \geq t$ and $S \geq t$, it means that $\exists \vec{C'_j} = 0$ and $\exists \vec{C_i} = 0$ after Step 15. Due to the permutation, $\exists \vec{C'_m} = 0$ and $\exists \vec{C_n} = 0$ holds after Step 16. Since $u' = u$, it follows that $([[k']], \mathsf{AES}_{K'}(1)) = ([[k]], \mathsf{AES}_K(1))$ in Step 17. Combining the previous two statements, we can conclude that $\exists B'_i, B_i : B'_i = B_i$ in Step 18 and thus $K' = K$. □

■

Appendix D

Correctness Proof of Partially Malicious Key Release Protocol

In this section, we will prove that the protocol as depicted in Figure 6.7 correctly computes functionality $f(\vec{p}, \perp) = (K \cdot \Big(\Sigma(\mathsf{LookUp}(\vec{p}, [[\mathbf{T}_u]])) \geq t\Big), \perp)$ as denoted in Equation A.1. First, in Section D.1 we will provide a proof for the correctness of $f_1(\vec{p}, \perp) = K \cdot \Big(\Sigma(\mathsf{LookUp}(\vec{p}, [[\mathbf{T}_u]])) \geq t\Big)$, in other words, the output of functionality f for the sensor device. Then in Section A.2, we will give a correctness proof for $f_2(\vec{p}, \perp) = \perp$, in other words, the output of functionality f for the verification server.

D.1 Sensor Device

We will prove that the protocol as depicted in Figures 6.7 correctly computes function $f_1(\vec{p}, \perp) = K \cdot \Big(\Sigma(\mathsf{LookUp}(\vec{p}, [[\mathbf{T}_u]])) \geq t\Big)$.

Proof. In case the sensor device is Party 1 and the verification server Party 2, let functionality $g_{x,y}(\vec{p}, \perp)$ be the intermediate functionality for Party x at Step y resembling the protocol in Figure 6.7. We will prove that at the end of a valid protocol run, $g_{1,32}(\vec{p}, \perp) = f_1(\vec{p}, \perp)$ applies.

Step 1-13: Since Steps 1-13 have no influence on the correctness of functionality $g_{x,y}(\vec{p}, \perp)$, we assume these steps are carried out correctly in the semi-honest model. More specifically, we assume that the sensor correctly captures a biometric identifier and it correctly constructs a valid template using ElGamal, which keeps the elements intact, similar to the work of Peeters et al. [56]. Also, we assume the sensor correctly chooses k and computes $\mathsf{AES}_K(1)$ and signatures over m and n. On the server side, we assume that m, n, $\sigma(m)$ and $\sigma(n)$ are correctly stored in the database and correctly fetched from the database upon later retrieval.

Step 14-15: At this point, we can start defining functionality g as follows:

$$g_{1,15}(\vec{p}, \perp) = \begin{cases} \perp & \text{if } \mathsf{Verify}(m, \sigma(m)) = \text{true} \wedge \ \mathsf{Verify}(n, \sigma(n)) = \text{true} \\ \perp & \text{otherwise} \end{cases} \tag{D.1}$$

Both verifications should be successful, otherwise the sensor aborts the protocol and the output for the sensor is $\perp$. Observe that at this point, the output for $g_{1,15}$ is always $\perp$, since no computations have been made yet.

Step 16: The sensor performs a look up procedure between $\vec{p}$ and $[[\mathbf{T}_u]]$, so can define g at this point as follows:

$$g_{1,16}(\vec{p}, \perp) = \begin{cases} \mathsf{LookUp}(\vec{p}, [[\mathbf{T}_u]]) & \text{if } \mathsf{Verify}(m, \sigma(m)) = \text{true} \wedge \ \mathsf{Verify}(n, \sigma(n)) = \text{true} \\ \perp & \text{otherwise} \end{cases} \tag{D.2}$$

However, since the result of the look up procedure yields a vector with encrypted similarity scores, we write $g_{1,16}$ as follows:

$$g_{1,16}(\vec{p},\perp) = \begin{cases} [[\mathsf{LookUp}(\vec{p},\mathbf{T}_u)]] & \text{if } \mathsf{Verify}(m,\sigma(m)) = \text{true} \wedge\ \mathsf{Verify}(n,\sigma(n)) = \text{true} \\ \perp & \text{otherwise} \end{cases} \tag{D.3}$$

Step 17: Since $[[S]]$ is the sum of all $[[s_i]]$ and thus $[[S]] = [[\Sigma(\mathsf{LookUp}(\vec{p},[[\mathbf{T}_u]]))]]$, we can write:

$$g_{1,17}(\vec{p},\perp) = \begin{cases} [[\Sigma(\mathsf{LookUp}(\vec{p},\mathbf{T}_u))]] & \text{if } \mathsf{Verify}(m,\sigma(m)) = \text{true} \wedge\ \mathsf{Verify}(n,\sigma(n)) = \text{true} \\ \perp & \text{otherwise} \end{cases} \tag{D.4}$$

Step 18: $[[S]]$ is sent to the server and thus we can write:

$$g_{2,18}(\vec{p},\perp) = \begin{cases} [[\Sigma(\mathsf{LookUp}(\vec{p},\mathbf{T}_u))]] & \text{if } \mathsf{Verify}(m,\sigma(m)) = \text{true} \wedge\ \mathsf{Verify}(n,\sigma(n)) = \text{true} \\ \perp & \text{otherwise} \end{cases} \tag{D.5}$$

Step 19/20a: Recall that compare function Compare$''$ checks if $[[S \geq t]]$ by computing $[[S - t - i]]$ for each $0 \leq i \leq \mathsf{max}(\mathbb{S}) - t$ without blinding factor r_i. Therefore we can define the following function for the sensor and server in Steps 19 and 20a:

$$g_{\{1,2\},\{19,20a\}}(\vec{p},\perp) = \begin{cases} [[\Sigma(\mathsf{LookUp}(\vec{p},\mathbf{T}_u)) - t - i]] & \text{if } \mathsf{Verify}(m,\sigma(m)) = \text{true} \wedge\ \mathsf{Verify}(n,\sigma(n)) = \text{true} \\ \perp & \text{otherwise} \end{cases} \tag{D.6}$$

for $0 \leq i \leq \mathsf{max}(\mathbb{S}) - t$.

Step 20b: For the server side of the compare function, we multiply the blinding factor r_i with the outcome of $g_{\{1,2\},\{19,20a\}}$ and thus we can define the following function:

$$g_{2,20b}(\vec{p},\perp) = \begin{cases} [[r_i(\Sigma(\mathsf{LookUp}(\vec{p},\mathbf{T}_u)) - t - i)]] & \text{if } \mathsf{Verify}(m,\sigma(m)) = \text{true} \wedge\ \mathsf{Verify}(n,\sigma(n)) = \text{true} \\ \perp & \text{otherwise} \end{cases} \tag{D.7}$$

for $0 \leq i \leq \mathsf{max}(\mathbb{S}) - t$ and $r_i \in_R \mathbb{Z}_q$.

See the work of Peeters et al. [56] for the correctness proof of this step.

Step 21: The server sends $[[\vec{C}]]$ to the sensor and thus we write:

$$g_{1,21}(\vec{p},\perp) = \begin{cases} [[r_i(\Sigma(\mathsf{LookUp}(\vec{p},\mathbf{T}_u)) - t - i)]] & \text{if } \mathsf{Verify}(m,\sigma(m)) = \text{true} \wedge\ \mathsf{Verify}(n,\sigma(n)) = \text{true} \\ \perp & \text{otherwise} \end{cases} \tag{D.8}$$

for $0 \leq i \leq \mathsf{max}(\mathbb{S}) - t$ and $r_i \in_R \mathbb{Z}_q$.

Step 22:

$$g_{\{1,2\},22}(\vec{p},\perp) = \begin{cases} [[r_i(\Sigma(\mathsf{LookUp}(\vec{p},\mathbf{T}_u)) - t - i)]] & \text{if } \mathsf{Verify}(m,\sigma(m)) = \text{true} \wedge\ \mathsf{Verify}(n,\sigma(n)) = \text{true} \\ & \wedge\ \mathsf{ZK\text{-}Proof}([[\vec{C''}]],[[\vec{C}]]) = \text{true} \\ \perp & \text{otherwise} \end{cases} \tag{D.9}$$

for $0 \leq i \leq \mathsf{max}(\mathbb{S}) - t$ and $r_i \in_R \mathbb{Z}_q$.

Step 23: Permutation function π is applied, so we can write:

$$g_{2,23}(\vec{p},\perp) = \begin{cases} \pi([[r_i(\Sigma(\mathsf{LookUp}(\vec{p},\mathbf{T}_u)) - t - i)]]) & \text{if } \mathsf{Verify}(m,\sigma(m)) = \text{true} \wedge\ \mathsf{Verify}(n,\sigma(n)) = \text{true} \\ & \wedge\ \mathsf{ZK\text{-}Proof}([[\vec{C''}]],[[\vec{C}]]) = \text{true} \\ \perp & \text{otherwise} \end{cases} \tag{D.10}$$

for $0 \leq i \leq \mathsf{max}(\mathbb{S}) - t$ and $r_i \in_R \mathbb{Z}_q$.

Step 24: The server sends $[[\vec{C'}]]$ to the sensor, and thus we can define the following functionality:

$$g_{1,24}(\vec{p}, \perp) = \begin{cases} \pi([[r_i(\Sigma(\mathsf{LookUp}(\vec{p}, \mathbf{T}_u)) - t - i)]]) & \text{if } \mathsf{Verify}(m, \sigma(m)) = \text{true} \wedge \ \mathsf{Verify}(n, \sigma(n)) = \text{true} \\ & \wedge\ \mathsf{ZK\text{-}Proof}([[\vec{C''}]], [[\vec{C}]]) = \text{true} \\ \perp & \text{otherwise} \end{cases} \tag{D.11}$$

for $0 \leq i \leq \mathsf{max}(\mathbb{S}) - t$ and $r_i \in_R \mathbb{Z}_q$.

Step 25:

$$g_{\{1,2\},25}(\vec{p}, \perp) = \begin{cases} \pi([[r_i(\Sigma(\mathsf{LookUp}(\vec{p}, \mathbf{T}_u)) - t - i)]]) & \text{if } \mathsf{Verify}(m, \sigma(m)) = \text{true} \wedge \ \mathsf{Verify}(n, \sigma(n)) = \text{true} \\ & \wedge\ \mathsf{ZK\text{-}Proof}([[\vec{C''}]], [[\vec{C}]]) = \text{true} \\ & \wedge\ \mathsf{ZK\text{-}Proof}([[\vec{C}]], [[\vec{C'}]]) = \text{true} \\ \perp & \text{otherwise} \end{cases} \tag{D.12}$$

for $0 \leq i \leq \mathsf{max}(\mathbb{S}) - t$ and $r_i \in_R \mathbb{Z}_q$.

Step 26/27: The sensor and server multiply the result of Step 23 with $[[k]]$, which has been fetched from the database and sent to the server in Steps 12 and 13. We can define the following functionalities for the sensor and server:

$$g_{\{1,2\},\{26,27\}}(\vec{p}, \perp) = \begin{cases} \pi([[r_i(\Sigma(\mathsf{LookUp}(\vec{p}, \mathbf{T}_u)) - t - i)]]) \cdot [[k]] & \text{if } \mathsf{Verify}(m, \sigma(m)) = \text{true} \\ & \wedge\ \mathsf{Verify}(n, \sigma(n)) = \text{true} \\ & \wedge\ \mathsf{ZK\text{-}Proof}([[\vec{C''}]], [[\vec{C}]]) = \text{true} \\ & \wedge\ \mathsf{ZK\text{-}Proof}([[\vec{C}]], [[\vec{C'}]]) = \text{true} \\ \perp & \text{otherwise} \end{cases} \tag{D.13}$$

for $0 \leq i \leq \mathsf{max}(\mathbb{S}) - t$ and $r_i \in_R \mathbb{Z}_q$.

Step 28: The server performs partial decryption function D_1 and so we can define g as follows:

$$g_{2,28}(\vec{p}, \perp) = \begin{cases} \pi([r_i(\Sigma(\mathsf{LookUp}(\vec{p}, \mathbf{T}_u)) - t - i)]) \cdot [k] & \text{if } \mathsf{Verify}(m, \sigma(m)) = \text{true} \\ & \wedge\ \mathsf{Verify}(n, \sigma(n)) = \text{true} \\ & \wedge\ \mathsf{ZK\text{-}Proof}([[\vec{C''}]], [[\vec{C}]]) = \text{true} \\ & \wedge\ \mathsf{ZK\text{-}Proof}([[\vec{C}]], [[\vec{C'}]]) = \text{true} \\ \perp & \text{otherwise} \end{cases} \tag{D.14}$$

for $0 \leq i \leq \mathsf{max}(\mathbb{S}) - t$ and $r_i \in_R \mathbb{Z}_q$.

Step 29: The server sends $[\vec{B}]$ to the sensor, and so we can define g at this point as:

$$g_{1,29}(\vec{p}, \perp) = \begin{cases} \pi([r_i(\Sigma(\mathsf{LookUp}(\vec{p}, \mathbf{T}_u)) - t - i)]) \cdot [k] & \text{if } \mathsf{Verify}(m, \sigma(m)) = \text{true} \\ & \wedge\ \mathsf{Verify}(n, \sigma(n)) = \text{true} \\ & \wedge\ \mathsf{ZK\text{-}Proof}([[\vec{C''}]], [[\vec{C}]]) = \text{true} \\ & \wedge\ \mathsf{ZK\text{-}Proof}([[\vec{C}]], [[\vec{C'}]]) = \text{true} \\ \perp & \text{otherwise} \end{cases} \tag{D.15}$$

for $0 \leq i \leq \mathsf{max}(\mathbb{S}) - t$ and $r_i \in_R \mathbb{Z}_q$.

Step 30:

$$g_{\{1,2\},30}(\vec{p},\perp) = \begin{cases} \pi([r_i(\Sigma(\mathsf{LookUp}(\vec{p},\mathbf{T}_u)) - t - i)]) \cdot [k] & \text{if } \mathsf{Verify}(m,\sigma(m)) = \text{true} \\ & \wedge\ \mathsf{Verify}(n,\sigma(n)) = \text{true} \\ & \wedge\ \mathsf{ZK\text{-}Proof}([[\vec{C''}]],[[\vec{C}]]) = \text{true} \\ & \wedge\ \mathsf{ZK\text{-}Proof}([[\vec{C}]],[[\vec{C'}]]) = \text{true} \\ & \wedge\ \mathsf{ZK\text{-}Proof}([[\vec{B}]],[\vec{B}]) = \text{true} \\ \perp & \text{otherwise} \end{cases} \tag{D.16}$$

for $0 \leq i \leq \mathsf{max}(\mathbb{S}) - t$ and $r_i \in_R \mathbb{Z}_q$.

Step 31: The sensor performs decryption function D_2, and so we can define g at this point as follows:

$$g_{1,31}(\vec{p},\perp) = \begin{cases} \pi(r_i(\Sigma(\mathsf{LookUp}(\vec{p},\mathbf{T}_u)) - t - i)) \cdot k & \text{if } \mathsf{Verify}(m,\sigma(m)) = \text{true} \\ & \wedge\ \mathsf{Verify}(n,\sigma(n)) = \text{true} \\ & \wedge\ \mathsf{ZK\text{-}Proof}([[\vec{C''}]],[[\vec{C}]]) = \text{true} \\ & \wedge\ \mathsf{ZK\text{-}Proof}([[\vec{C}]],[[\vec{C'}]]) = \text{true} \\ & \wedge\ \mathsf{ZK\text{-}Proof}([[\vec{B}]],[\vec{B}]) = \text{true} \\ \perp & \text{otherwise} \end{cases} \tag{D.17}$$

for $0 \leq i \leq \mathsf{max}(\mathbb{S}) - t$ and $r_i \in_R \mathbb{Z}_q$.

However, we can reduce $g_{1,31}$ to the following function, since all the signature verifications and zero-knowledge proofs should have succeeded in order to reach the protocol at this point and thus there is no case distinction anymore:

$$g_{1,31}(\vec{p},\perp) = \pi(r_i(\Sigma(\mathsf{LookUp}(\vec{p},\mathbf{T}_u)) - t - i))) \cdot k \tag{D.18}$$

Step 32: In the protocol, the sensor checks statement $\left(\exists B_i : \mathsf{AES\text{-}D}_{B_i}(\mathsf{AES}_K(1)) = 1\right)$ and in case it is true, it extracts B_i as key K. Moreover, we can remove permutation function π, because it does not have influence on the result of the key extraction. Therefore, we can write:

$$g_{1,31}(\vec{p},\perp) = K \cdot \left(\exists B_i : \mathsf{AES\text{-}D}_{B_i}(\mathsf{AES}_K(1)) = 1\right) \tag{D.19}$$

where $B_i = K \cdot r_i(\Sigma(\mathsf{LookUp}(\vec{p},\mathbf{T}_u)) - t - i)$ for $0 \leq i \leq \mathsf{max}(\mathbb{S}) - t$.

Observe that $g_{1,31}$ is similar to $g_{1,22}(\vec{p},\perp)$ in the correctness proof of the semi-honest key release protocol of Appendix A. We will therefore rely on the corresponding correctness proof, so we can conclude that $g_{1,31}(\vec{p},\perp) = f_1(\vec{p},\perp)$ holds. ■

D.2 Server

Similar to the semi-honest key release protocol, the proof that the protocol as depicted in Figure 6.7 correctly computes function $f_2(\vec{p},\perp) = \perp$ is trivial, as the protocol does not have any output for the server. Thus we can conclude that $f_2(\vec{p},\perp) = \perp$.

Appendix E

Security Proof of Partially Malicious Key Release Protocol

As mentioned before in Section 6.2.3, in order to prove the key release protocol from Section 6 is secure in the partially malicious model as defined in Definition 6, we first prove the corresponding protocol is secure in the semi-honest model. Hence we consider the following two cases:

(a) The server is honest and the sensor device is compromised by a semi-honest attacker

(b) The sensor device is honest and the server is compromised by a semi-honest attacker

Subsequently, giving an argumentative proof using the GMW compiler, we will prove security of the corresponding key release protocol in the partially malicious model, i.e. security against a semi-honest sensor and a malicious server. Hence, we consider the following case:

(c) The sensor device is compromised by a semi-honest attacker and the server is compromised by a malicious attacker

Recall from Definition 3 that a deterministic function f securely computes protocol π in the semi-honest model if there exist probabilistic polynomial-time algorithms, S_1 and S_2, such that:

$$\{S_1(x, f_1(x, y))\}_{x,y\in\{0,1\}^*} \stackrel{c}{\equiv} \{\mathsf{VIEW}_1^{\pi}(x, y)\}_{x,y\in\{0,1\}^*} \tag{2.3}$$

$$\{S_2(y, f_2(x, y))\}_{x,y\in\{0,1\}^*} \stackrel{c}{\equiv} \{\mathsf{VIEW}_2^{\pi}(x, y)\}_{x,y\in\{0,1\}^*} \tag{2.4}$$

where $|x| = |y|$ and $f_1(x, y)$ and $f_2(x, y)$ are the outputs of Party 1 and 2, respectively.

In other words, there exists an algorithm or simulator S_i for $i \in \{1, 2\}$ in the ideal model execution of π that is, given the input and the output of S_i, computationally indistinguishable from the view of Party i in the real model execution of π [45].

As mentioned in Appendix B, the sensor device has private input $x = \vec{p}$ and the server has no private input, hence we denote its input by $y = \perp$. We define function f as follows:

$$f(\vec{p}, \perp) = (K \cdot \Big(\Sigma(\mathsf{LookUp}(\vec{p}, [[\mathbf{T}_u]])) \geq t\Big), \perp)$$

The sensor obtains a key based on whether or not the biometric identifier presented to it was successfully verified. The server has no output.

E.1 Case (a)

Proof. Assume that the sensor device is Party 1 and let $f_1'(\vec{p}, \perp)$ be the functionality that is computed by S_1. Note that if the term $\Sigma(\mathsf{LookUp}(\vec{p}, [[\mathbf{T}_u]])) \geq t$ is 1 (true), then $f_1(\vec{p}, \perp) = K$, else $f_1(\vec{p}, \perp) = 0$ (false). Hence, for the verification procedure from Step 10 onwards, we need to distinguish between two cases:

1. $f_1(\vec{p}, \perp) = K$
2. $f_1(\vec{p}, \perp) = 0$

In case $f_1(\vec{p}, \perp) = K$, we can construct the following subproof:

Subproof. For S_1, in Figure 6.7, replace messages $m = (u, [[\mathbf{T}_u]]), n = (u, [[k]], \mathsf{AES}_K(1)), u$ and $[[S]]$ in Steps 8, 11 and 18 with $m' = (u', [[\mathbf{T}'_u]]), n = (u', [[k']], \mathsf{AES}_{K'}(1)), u'$ and $[[S']]$, respectively, such that $S' = \mathsf{max}(\mathbb{S})$. Moreover, replace challenges c for the zero-knowledge proofs in Steps 22, 25 and 30 with challenges c'. Since the challenges do not have any influence on S', they can be chosen uniformly at random. Now $[[S' \geq t]]$ holds and thus $\exists B'_i : B'_i = K$. We can therefore state that $f'_1(\vec{p}, \perp) = K = f_1(\vec{p}, \perp)$. The sensor device knows $\mathsf{max}(\mathbb{S})$, because it is a public value. Also, both the sensor device and the verification server have encryption keys pk_{shared} and pk_{sv}, so the sensor is able to encrypt S'. We can state now that S_1 is a polynomial-time algorithm. □

In case $f(\vec{p}, \perp) = 0$, we can construct the following subproof:

Subproof. For S_1, in Figure 5.2, replace messages $m = (u, [[\mathbf{T}_u]]), n = (u, [[k]], \mathsf{AES}_K(1)), u$ and $[[S]]$ in Steps 8, 11 and 18 with $m' = (u', [[\mathbf{T}'_u]]), n = (u', [[k']], \mathsf{AES}_{K'}(1)), u'$ and $[[S']]$, respectively, such that $S' = \mathsf{min}(\mathbb{S})$. Moreover, replace challenges c for the zero-knowledge proofs in Steps 22, 25 and 30 with challenges c'. Since the challenges do not have any influence on S', they can be chosen uniformly at random. Now $[[S' < t]]$ holds and thus $\nexists B'_i : B'_i = K$. We can therefore state that $f'_1(\vec{p}, \perp) = 0 = f_1(\vec{p}, \perp)$. Similar to the previous case, the sensor device knows $\mathsf{min}(\mathbb{S})$ and is able to encrypt S' and thus S_1 is a polynomial-time algorithm. □

For both cases, we can state that ElGamal is secure under a chosen plaintext attack [62] and that the sensor does not possess sk_{shared} due to the threshold encryption system, thus S_1 is computationally indistinguishable from the real model view. We can therefore conclude that Equation 2.3 holds and thus a compromised sensor device is not able to learn anything during the protocol run, except from its own input and output value. ■

E.2 Case (b)

Proof. Assume that the server is Party 2 and let S_2 be the algorithm that computes f'_2. In Figure 6.7, replaces message $m = (u, [[\mathbf{T}_u]]), n = (u, [[k]], \mathsf{AES}_K(1)), [[\vec{C}]], [[\vec{C'}]]$ and $[\vec{B}]$ in Steps 13, 21, 24 and 29, with $m' = (u', [[\mathbf{T}'_u]]), n' = (u', [[k']], \mathsf{AES}_{K'}(1)), [[\vec{C}']], [[\vec{C''}]]$ and $[\vec{B}']$, respectively, such that $f'_2(\vec{p}, \perp) = f_2(\vec{p}, \perp) = \perp$. Note that messages $[[\vec{C}']], [[\vec{C''}]]$ have already been defined previously in Steps 24 and 20b of Figure 6.7. However, in this context, they mean the replacement messages of Steps 21 and 24 in the security proof of the partially malicious key release protocol. Moreover, replace commitments Com_{Step22}, Com_{Step25} and Com_{Step30} and response messages Resp_{Step22}, Resp_{Step25} and Resp_{Step30} that the server sends in the zero-knowledge proofs of Steps 22, 25 and 30 with Com'_{Step22}, $\mathsf{Com}'_{Step25}, \mathsf{Com}'_{Step30}, \mathsf{Resp}'_{Step22}, \mathsf{Resp}'_{Step25}$ and Resp'_{Step30}, such that $f'_2(\vec{p}, \perp) = f_2(\vec{p}, \perp) = \perp$. Since the verification server has no output ($\perp$), we can choose $u', \mathbf{T}_u, K', \vec{C}', \vec{C''}, \vec{B}$, Com'_{Step22}, $\mathsf{Com}'_{Step25}, \mathsf{Com}'_{Step30}, \mathsf{Resp}'_{Step22}, \mathsf{Resp}'_{Step25}$ and Resp'_{Step30} uniformly at random, which can be done in polynomial time. Observe that choosing $u', \mathbf{T}_u, K', \vec{C}', \vec{C''}, \vec{B}, \mathsf{Com}'_{Step22}$, $\mathsf{Com}'_{Step25}, \mathsf{Com}'_{Step30}, \mathsf{Resp}'_{Step22}, \mathsf{Resp}'_{Step25}$ and Resp'_{Step30} such that all the signature verifications and zero-knowledge proofs will succeed or one or more fails is not important, since in all cases $f'_2(\vec{p}, \perp) = f_2(\vec{p}, \perp) = \perp$ holds. In the former case, $f'_2(\vec{p}, \perp)$ will output $\perp$, since there is no output for the server and in the latter case, the sensor aborts the protocol prematurely, so the server outputs $\perp$ as well. Again, since ElGamal is secure under a chosen plaintext attack [62], and the server does not possess sk_{shared} due to the threshold encryption system, S_2 is computationally indistinguishable from the real model view. We can therefore conclude that Equation 2.4 holds and a compromised verification server is not able to learn anything during the protocol, except from its own input and output value. ■

E.3 Case (c)

Proof Sketch. Let the partially malicious key release protocol be defined as π. Recall from Section 6.1.1 that the GMW compiler consists of three stages: the Input Commitment protocol, the Coin Generation phase and the Protocol Emulation.

First of all, the Input Commitment protocol is not needed in this setting. The sensor device is not malicious and thus does not need to commit to its input. The server might be malicious, but does not have any input to commit to. For the Coin Generation phase, the Augmented Coin Tossing

protocol is used. Since the sensor is not malicious, it does not need to toss random coins. We treat the random blinding factors as generated in the comparison function through the Augmented Coin Tossing protocol as the random coins used by the server.

Note that since the sensor is semi-honest in the worst-case scenario, we rely for its steps on the security proof of Section E.1 and thus we will only regard the steps taken by the server.

Now for the protocol emulation, we prove each step of the server secure in the malicious model as follows.

Step 9: No direct proof is needed for the table storage, since the sensor device will detect later in Steps 14 and 15 malicious behavior of such kind.

Step 12-15: Steps 12-15 are trivial to prove, since they are basically an implementation of two Schnorr Signature verification schemes. We therefore rely on the works of Abdalla et al. [3] and Ohta et al. [53], who proved that the Schnorr Signature Scheme is secure against existential forgery under a chosen message attack in the random oracle model, relying on the hardness of the discrete logarithm problem and thus secure in a setting where the server is malicious and the sensor (semi-)honest.

Step 19-22: Let the server be Party 1 and the sensor be Party 2 in the current protocol emulation of the GMW compiler. Recall from Section 6.1.1 that the server has input $\alpha = (\alpha_1, \alpha_2, \alpha_3)$ to function $f(\alpha)$ in the Authenticated Computation Protocol, where $\alpha_1 = (u, \rho^1)$ is the output of the Input Commitment protocol for secret input value u, $\alpha_2 = (r^1, \omega^1)$ the output of the Augmented Coin Tossing protocol and α_3 the list of messages Party 1 and 2 have received earlier in the protocol. The sensor has input $\beta = (\gamma^1, \delta^1, \alpha_3)$ to the Authenticated Computation Protocol, where $\gamma^1 = \mathsf{Com}(u, \rho^1)$ and $\delta^1 = \mathsf{Com}(r^1, \omega^1)$ denote the respective outputs of the Input Commitment and Coin Tossing protocol for Party 2.

The current protocol emulation step is performed $0 \leq i \leq j$ times where $j = \mathsf{max}(\mathbb{S}) - t$. As random coin r^1, blinding factor r_i received from the Authenticated Coin Tossing protocol in Step 20b is used, thus $\alpha_{2i} = (r_i^1, \omega_i^1)$ and $\delta_i^1 = \mathsf{Com}(r_i^1, \omega^1)$. The list of earlier received messages can be denoted by $\alpha_3 = ((m, n, \sigma(m), \sigma(n)), u, [[S]])$. We can define α_i and β_i now as follows:

$$\alpha_i = \big((\bot, \bot), (r_i^1, \omega_i^1), ((m, n, \sigma(m), \sigma(n)), u, [[S]])\big) \tag{E.1}$$

$$\beta_i = \big(\bot, \mathsf{Com}(r_i^1, \omega^1), ((m, n, \sigma(m), \sigma(n)), u, [[S]])\big) \tag{E.2}$$

Additionally, we set $f(\alpha_i)$ to $g_{2,20b}$ from Section D.1. Hence,

$$\begin{aligned} f(\alpha_i) &= \begin{cases} [[r_i(\Sigma(\mathsf{LookUp}(\vec{p}, \mathbf{T}_u)) - t - i)]] & \text{if } \mathsf{Verify}(m, \sigma(m)) = \text{true} \wedge\ \mathsf{Verify}(n, \sigma(n)) = \text{true} \\ \bot & \text{otherwise} \end{cases} \\ &= \begin{cases} [[r_i(\Sigma(S - t - i)]] & \text{if } \mathsf{Verify}(m, \sigma(m)) = \text{true} \wedge\ \mathsf{Verify}(n, \sigma(n)) = \text{true} \\ \bot & \text{otherwise} \end{cases} \end{aligned} \tag{E.3}$$

In the current context, we can define $h(\alpha_i)$ as follows:

$$h(\alpha_i) = h(\alpha_1, \alpha_{2i}, \alpha_3) = (\mathsf{Com}(u, \rho^1), \mathsf{Com}(r_i^1, \omega^1), \alpha_3) = (\bot, \mathsf{Com}(r_i^1, \omega^1), \alpha_3) \tag{E.4}$$

Now in Step 22, the server and sensor engage in a secure implementation of the Authenticated Computation Protocol using their inputs α_i and β_i and $f(\alpha_i)$ and $h(\alpha_i)$ [28].

Recall from Equation 6.6 that the zero-knowledge protocol in the current step is defined as:

$$\mathsf{PK}\{(\psi, \omega) : c_1 = c_1''^{\psi} \wedge c_2 = c_2''^{\psi} \wedge \mathsf{Com}(r_i, \rho_i) = g^{\omega} h^{\psi}\} \tag{6.6}$$

To avoid duplicate notations, we have replaced α and β in Equation 6.6 by ψ and ω, respectively. We assume that the notions of g and h can be learned from their context.

Now observe from Appendix D that

$$g_{\{1,2\},\{19,20a\}}(\vec{p},\perp) = \begin{cases} [[\Sigma(\mathsf{LookUp}(\vec{p},\mathbf{T}_u)) - t - i]] & \text{if } \mathsf{Verify}(m,\sigma(m)) = \text{true} \wedge\ \mathsf{Verify}(n,\sigma(n)) = \text{true} \\ \perp & \text{otherwise} \end{cases}$$
$$= \begin{cases} [[r_i(\Sigma(S - t - i)]] & \text{if } \mathsf{Verify}(m,\sigma(m)) = \text{true} \wedge\ \mathsf{Verify}(n,\sigma(n)) = \text{true} \\ \perp & \text{otherwise} \end{cases} \tag{D.6}$$

for $0 \leq i \leq j$.

Moreover, observe from Step 22 in Section 6.2.2 that the first component of ElGamal encryption $f(\alpha_i)$ equals c_1 and the second component equals c_2, since $f(\alpha_i) = [[\vec{C}_i]$. Similarly, the first component of ElGamal encryption $g_{\{1,2\},\{19,20a\}}$ equals c_1'' and the second component equals c_2'', since $g_{\{1,2\},\{19,20a\}} = [[C'']]$. Since π is secure in the partially malicious model up to Step 19/20a, $f(\alpha_i)$ can safely be computed from $g_{\{1,2\},\{19,20a\}}$. Proving that $\exists\psi : c_1 = c_1''^{\psi} \wedge c_2 = c_2''^{\psi}$ now is essentially equal to proving that $\exists x : f(\alpha_i) = f(x)$ in the Authenticated Computation protocol.

Additionally, in a more obvious way, as can be observed from Equation E.4, proving that $\exists\psi,\omega :$ $\mathsf{Com}(r_i,\rho_i) = g^{\omega}h^{\psi}$ is essentially equal to proving that $\exists x : h(\alpha_i) = h(x)$ and $h(\alpha_i) = \beta$ in the Authenticated Computation protocol. Yet observe that α_3 is not included in this part of the zero-knowledge proof, but has already been included in the proof of $f(\alpha_i)$. Also, proving $h(\alpha_i)$ in combination with $f(\alpha_i)$ guarantees that the server used the correct blinding factor.

Using AND-compositions to concatenate the previous two statements and given that the zero-knowledge proof of Equation 6.6 is honest-verifier zero-knowledge [36] (i.e. secure against a semi-honest sensor and malicious server), we can conclude that Equation 6.6 is a secure implementation of the zero-knowledge proof-of-knowledge of the current invocation of the Authenticated Computation protocol and thus π is secure in the partially malicious model up to this point.

Step 23-25: Again, let the server be Party 1 and the sensor be Party 2 in the current protocol emulation step of the GMW compiler. We can define α_i and β_i now as follows:

$$\alpha = \Big((\perp,\perp),\{(r_i^1,\omega_i^1)\},((m,n,\sigma(m),\sigma(n)),u,[[S]],[[\vec{C}]])\Big) \tag{E.5}$$

$$\beta = \Big(\perp,\{\mathsf{Com}(r_i^1,\omega_i^1)\},((m,n,\sigma(m),\sigma(n)),u,[[S]],[[\vec{C}]])\Big) \tag{E.6}$$

Note that in the security proof of Step 19-22, we expressed α and β for each i separately, but in Step 23-25, since we only need to invoke the Authenticated Computation protocol once, the second arguments of α and β are now sets of the random coins and their commitments, respectively.

We set $f(\alpha)$ to $g_{2,28}$ from Section D.1, hence,

$$f(\alpha) = \begin{cases} \pi([[r_i(\Sigma(\mathsf{LookUp}(\vec{p},\mathbf{T}_u)) - t - i)]]) & \text{if } \mathsf{Verify}(m,\sigma(m)) = \text{true} \wedge\ \mathsf{Verify}(n,\sigma(n)) = \text{true} \\ & \wedge\ \mathsf{ZK\text{-}Proof}([[\vec{C''}]],[[\vec{C}]]) = \text{true} \\ \perp & \text{otherwise} \end{cases} \tag{E.7}$$

for $0 \leq i \leq \mathsf{max}(\mathbb{S}) - t$ and $r_i \in_R \mathbb{Z}_q$.

We define $h(\alpha)$ as follows:

$$h(\alpha) = (\perp,\{\mathsf{Com}(r_i^1,\omega_i^1)\},\alpha_3) \tag{E.8}$$

Now in Step 25, the server and sensor engage in a secure implementation of the Authenticated Computation Protocol using their inputs α and β and $f(\alpha)$ and $h(\alpha)$ [28].

Recall from Equation 6.8 that the zero-knowledge protocol in the current step is defined as:

$$\mathsf{PK}\{(\phi,x) : \mathsf{D}_x([[\vec{C'}]]) = \mathsf{D}_x([[\vec{C'}]]_{\phi^{-1}(i)})\}\} \tag{6.7}$$

for permutation function ϕ and decryption function D with secret key x.

Now observe from Appendix D that

$$g_{2,20b}(\vec{p},\perp) = \begin{cases} [[r_i(\Sigma(\mathsf{LookUp}(\vec{p},\mathbf{T}_u)) - t - i)]] & \text{if } \mathsf{Verify}(m,\sigma(m)) = \text{true} \wedge\ \mathsf{Verify}(n,\sigma(n)) = \text{true} \\ \perp & \text{otherwise} \end{cases} \tag{D.7}$$

for $0 \leq i \leq \mathsf{max}(\mathbb{S}) - t$ and $r_i \in_R \mathbb{Z}_q$.

Moreover, observe from Step 25 in Section 6.2.2 that $f(\alpha) = [[\vec{C'}]]$ and that $g_{2,20b} = [[\vec{C}]]$. Since π is secure in the partially malicious model up to Step 22, $f_i(\alpha)$ can safely be computed. Proving that $(\phi, x) : \mathsf{D}_x([[\vec{C'}]]) = \mathsf{D}_x([[\vec{C'}]]_{\phi^{-1}(i)})$ now is essentially equal to proving that $\exists x : f(\alpha) = f(x)$ in the Authenticated Computation protocol.

Now proving that $\exists x : h(\alpha) = h(x)$ is already implied by the proof of the blinding factor in Step 22. Since π is secure in the partially malicious model up to this point, we can state that $(\phi, x) : \mathsf{D}_x([[\vec{C'}]]) = \mathsf{D}_x([[\vec{C'}]]_{\phi^{-1}(i)})$ is enough to prove $\exists x : h(\alpha) = h(x)$.

Now given that the permutation proof of Furukawa [24] is honest-verifier zero-knowledge, we can conclude that Equation 6.7 is a secure implementation of the zero-knowledge proof-of-knowledge of the current invocation of the Authenticated Computation protocol and thus π is secure in the partially malicious model up to this point.

Step 26-30: Again, let the server be Party 1 and the sensor be Party 2 in the current protocol emulation step of the GMW compiler. The current protocol emulation step is performed $0 \leq i \leq j$ times where $j = \mathsf{max}(\mathbb{S}) - t$ for each element of $[\vec{B}]$. We can define α_i and β_i now as follows:

$$\alpha_i = \Big((\bot, \bot), (r_i^1, \omega_i^1), ((m, n, \sigma(m), \sigma(n)), u, [[S]], [[\vec{C}]], [[\vec{C'}]])\Big) \tag{E.9}$$

$$\beta_i = \Big(\bot, \mathsf{Com}(r_i^1, \omega_i^1), ((m, n, \sigma(m), \sigma(n)), u, [[S]], [[\vec{C}]], [[\vec{C'}]])\Big) \tag{E.10}$$

Additionally, we set $f(\alpha_i)$ to $g_{2,28}$ from Section D.1. Hence,

$$\begin{aligned} f(\alpha_i) &= \begin{cases} \pi([r_i(\Sigma(\mathsf{LookUp}(\vec{p}, \mathbf{T}_u)) - t - i)]) \cdot [k] & \text{if } \mathsf{Verify}(m, \sigma(m)) = \text{true} \\ & \quad \wedge\ \mathsf{Verify}(n, \sigma(n)) = \text{true} \\ & \quad \wedge\ \mathsf{ZK\text{-}Proof}([[\vec{C''}]], [[\vec{C}]]) = \text{true} \\ & \quad \wedge\ \mathsf{ZK\text{-}Proof}([[\vec{C}]], [[\vec{C'}]]) = \text{true} \\ \bot & \text{otherwise} \end{cases} \\ &= \begin{cases} \pi([r_i(S - t - i)]) \cdot [k] & \text{if } \mathsf{Verify}(m, \sigma(m)) = \text{true} \\ & \quad \wedge\ \mathsf{Verify}(n, \sigma(n)) = \text{true} \\ & \quad \wedge\ \mathsf{ZK\text{-}Proof}([[\vec{C''}]], [[\vec{C}]]) = \text{true} \\ & \quad \wedge\ \mathsf{ZK\text{-}Proof}([[\vec{C}]], [[\vec{C'}]]) = \text{true} \\ \bot & \text{otherwise} \end{cases} \end{aligned} \tag{E.11}$$

for $0 \leq i \leq \mathsf{max}(\mathbb{S}) - t$ and $r_i \in_R \mathbb{Z}_q$.

We define $h_i(\alpha_i)$ as follows:

$$h(\alpha_i) = (\bot, \mathsf{Com}(r_i^1, \omega^1), \alpha_3) \tag{E.12}$$

Now in Step 30, the server and sensor engage in a secure implementation of the Authenticated Computation Protocol using their inputs α_i and β_i and $f(\alpha_i)$ and $h(\alpha_i)$ [28].

Recall from Equation 6.8 that the zero-knowledge protocol in the current step is defined as:

$$\mathsf{PK}\{(\psi) : c_1' = c_1^{\psi} \wedge \mathsf{pk}_{sv} = g^{\psi} \mid \forall c' = [B_i], c = [[B_i]]\} \tag{6.8}$$

To avoid duplicate notations, we have replaced α in Equation 6.8 by ψ, respectively.

Now observe from Appendix D that

$$g_{\{1,2\},\{26,27\}}(\vec{p},\perp) = \begin{cases} \pi([[r_i(\Sigma(\mathsf{LookUp}(\vec{p},\mathbf{T}_u)) - t - i)]]) \cdot [[k]] & \text{if } \mathsf{Verify}(m,\sigma(m)) = \text{true} \\ & \wedge\ \mathsf{Verify}(n,\sigma(n)) = \text{true} \\ & \wedge\ \mathsf{ZK\text{-}Proof}([[\vec{C''}]],[[\vec{C}]]) = \text{true} \\ & \wedge\ \mathsf{ZK\text{-}Proof}([[\vec{C}]],[[\vec{C'}]]) = \text{true} \\ \perp & \text{otherwise} \end{cases}$$

$$= \begin{cases} \pi([[r_i(S - t - i)]]) \cdot [[k]] & \text{if } \mathsf{Verify}(m,\sigma(m)) = \text{true} \\ & \wedge\ \mathsf{Verify}(n,\sigma(n)) = \text{true} \\ & \wedge\ \mathsf{ZK\text{-}Proof}([[\vec{C''}]],[[\vec{C}]]) = \text{true} \\ & \wedge\ \mathsf{ZK\text{-}Proof}([[\vec{C}]],[[\vec{C'}]]) = \text{true} \\ \perp & \text{otherwise} \end{cases} \tag{D.13}$$

for $0 \leq i \leq \mathsf{max}(\mathbb{S}) - t$ and $r_i \in_R \mathbb{Z}_q$.

Moreover, observe from Step 30 in Section 6.2.2 that the first component of ElGamal encryption $f(\alpha_i)$ equals c_1' and the second component equals c_2', since $f(\alpha_i) = [B_i]$. Similarly, the first component of ElGamal encryption $g_{\{1,2\},\{26,27\}}$ equals c_1 and the second component equals c_2, since $g_{\{1,2\},\{26,27\}} = [[B_i]]$. Since π is secure in the partially malicious model up to Step 25 and the Step 26 is securely computed by the sensor device, $f_i(\alpha)$ can safely be computed. Proving that $\exists\psi : c_1' = c_1^{\psi}$ now is essentially equal to proving that $\exists x_i : f(\alpha_i) = f(x_i)$ in the Authenticated Computation protocol. Additionally proving in an AND composition that $\exists\psi : \mathsf{pk}_{sv} = g^{\psi}$ guarantees that $\psi = \mathsf{sk}_{sv}$.

Now proving that $\exists x_i : h(\alpha_i) = h(x_i)$ is already implied by the proof of the blinding factor in Step 22. Since π is secure in the partially malicious model up to Step 25, we can state that $\exists\psi : c_1' = c_1^{\psi}$ is enough to prove $\exists x_i : h(\alpha_i) = h_i(x_i)$.

Now given that the zero-knowledge proof of Equation 6.8 is honest-verifier zero-knowledge [36], we can conclude that Equation 6.8 is a secure implementation of the zero-knowledge proof-of-knowledge of the current invocation of the Authenticated Computation protocol and thus π is secure in the partially malicious model up to this point.

We can now conclude that the partially malicious key release protocol is secure against a semi-honest sensor device and a malicious server.

■

Appendix F

Runtime Experiments

	fs1a	**fs1b**	**fs1c**	**fs1d**	**fs1e**	**fs1f**
Enrollment:						
Setup	1.8456	1.5347	1.8908	2.1941	2.6037	2.312
Enrollment	88.5761	91.9781	93.5658	175.7503	175.087	176.4234
Total:	90.4217	93.5128	95.4566	177.9444	177.6907	178.7354
Verification:						
Capture	0.4758	0.4809	0.3884	1.4316	1.3352	1.1848
Fetch from table	0.0195	0.0183	0.0179	0.0457	0.0171	0.02
Signature verification	4.449	4.8746	4.6985	7.3043	7.9145	7.2093
Look up	0.0127	0.019	0.0172	0.0293	0.0214	0.0163
Add scores	5.2294	5.6536	5.5896	10.2699	10.7761	10.5156
Comparison	2849.4352	1499.7727	777.3194	2961.5491	1503.8536	769.775
ZK-Proof comparison	148.7538	81.2169	45.5136	156.0989	83.2564	43.5812
Permutation	42.0334	22.2378	11.523	43.9847	22.0011	11.4023
ZK-Proof permutation	404.9868	205.4843	109.0588	423.0849	202.9254	106.007
Add key K	54.5883	29.542	18.1032	56.1972	30.9924	15.7649
Partial decryption	19.8848	11.3825	5.7822	20.7464	10.9728	5.6115
ZK-Proof partial decryption	54.9224	30.5198	16.5595	56.0222	30.8033	17.7065
Full decryption	21.412	11.3816	5.7851	21.9924	11.8357	6.3229
Key check	1.1377	0.5677	0.3415	1.0118	0.6761	0.4074
Total communication overhead	39.4002	32.1603	26.092	57.7654	54.2816	49.3271
Total:	3646.741	1935.312	1026.7899	3817.5338	1971.6627	1044.8518

Table F.1: Run times of the different components of the partially malicious key release protocol for feature set fs1 (ms)

	fs2a	**fs2b**	**fs2c**	**fs2d**	**fs2e**	**fs2f**
Enrollment:						
Setup	1.7218	2.8872	2.55	3.5162	2.9249	1.8479
Enrollment	92.1256	90.6867	91.4687	171.7671	171.1643	171.5545
Total:	93.8474	93.5739	94.0187	175.2833	174.0892	173.4024
Verification:						
Capture	0.5744	0.4823	0.5554	2.4437	1.028	1.7941
Fetch from table	0.0241	0.0161	0.0219	0.0194	0.0211	0.027
Signature verification	4.396	4.686	5.1149	6.9889	7.4365	7.0922
Look up	0.0132	0.0249	0.022	0.0282	0.0172	0.03
Add scores	5.4552	5.7067	5.7542	10.3989	10.5432	10.4635
Comparison	2842.2064	1425.9282	740.9136	2753.6456	1422.3791	735.7225
ZK-Proof comparison	155.3518	78.0624	42.0602	149.2608	78.7346	41.8411
Permutation	41.9649	21.8443	10.983	41.6218	21.527	12.044
ZK-Proof permutation	409.8173	201.7364	103.5704	406.8189	195.1128	105.4092
Add key K	53.1873	31.9572	17.1545	56.2018	31.4675	17.8993
Partial decryption	20.5466	10.6537	5.681	19.851	11.0128	5.7891
ZK-Proof partial decryption	54.2205	29.7226	16.4224	54.7917	29.129	16.5104
Full decryption	21.22	11.2987	6.5984	21.0567	10.9265	6.1187
Key check	1.1131	0.5785	0.3651	1.1537	0.6528	0.2195
Total communication overhead	39.8203	31.0941	26.3846	55.0034	48.7786	46.1431
Total:	3649.9111	1853.7921	981.6016	3579.2845	1868.7667	1007.1037

Table F.2: Run times of the different components of the partially malicious key release protocol for feature set fs2 (ms)

	fs3a	**fs3b**	**fs3c**	**fs3d**	**fs3e**	**fs3f**
Enrollment:						
Setup	2.5552	1.9639	1.4991	1.9305	2.1568	2.121
Enrollment	57.6156	57.5395	57.8499	107.1507	102.8415	107.7588182
Total:	60.1708	59.5034	59.349	109.0812	104.9983	109.8798182
Verification:						
Capture	0.4294	0.5954	0.425	0.4939	0.4962	0.535727273
Fetch from table	0.0241	0.0243	0.0169	0.017	0.0194	0.011
Signature verification	3.7049	3.4573	3.3245	5.4318	5.0295	5.323454545
Look up	0.0263	0.0189	0.0138	0.0382	0.0185	0.019363636
Add scores	5.8676	5.7543	5.662	11.2627	10.3694	10.836
Comparison	1720.2473	917.8655	481.7604	1796.2368	876.3824	473.7690909
ZK-Proof comparison	97.0913	50.8254	27.142	99.3711	47.8511	26.669
Permutation	25.2305	13.262	7.2788	25.8592	12.9071	7.036909091
ZK-Proof permutation	237.1475	123.7099	67.3028	233.1225	122.0844	67.19954545
Add key K	34.1287	19.3752	8.8353	37.3154	19.8475	10.02063636
Partial decryption	12.3379	6.752	3.6347	12.4047	6.5552	3.554636364
ZK-Proof partial decryption	34.1335	20.1224	10.4375	34.7791	18.8365	10.55581818
Full decryption	13.3385	6.8135	3.7965	13.5358	6.4025	3.455181818
Key check	0.6412	0.3617	0.215	0.6865	0.2745	0.195
Total communication overhead	23.7751	20.7256	18.534	35.4853	31.5887	29.33609091
Total:	2208.1238	1189.6634	638.3792	2306.04	1158.6629	648.5174545

Table F.3: Run times of the different components of the partially malicious key release protocol for feature set fs3 (ms)

Appendix G

Installation Instructions

Install Instructions:

- `git clone https://github.com/Ruth1993/thesis.git`
- Install libscapi as on: `https://github.com/cryptobiu/libscapi/tree/dev`. If it does not compile, please try `https://github.com/Ruth1993/libscapi`.
- To check if the installation was successful, build and run the tests as on: `https://biulibscapi.readthedocs.io/en/latest/install.html`

For the installation of the partially malicious key release protocol, please carry out the additional steps:

- `cd ~/libscapi/include/primitives/Dlog.hpp`
- Change line 811 and/or line 813 (depending on the OS) to the correct file path or copy

  `NISTEC.txt`

  to the specified location. In case folders thesis and libscapi are located in the same folder, change line 811 or 813 to:

  `const string NISTEC\_PROPERTIES\_FILE = "../../libscapi/include/configFiles/NISTEC.txt"`
- `cd ../..`
- `make`

Compile:

- `~/thesis`
- `make`

Run: Open 2 terminals and run the protocols as follows:

- Semi-honest Key Release Protocol:
  - `./server`
    or
    `./server sh`
  - `./sensor`
    or
    `./sensor sh`
- Partially Malicious Key Release Protocol:
  - `./server mal`
  - `./sensor mal`

The code has been tested and found working on Ubuntu 16.04 with OpenSSL 1.0.2g.