A Framework for UC Secure Privacy Preserving Biometric Authentication using Efficient Functional Encryption

Johannes Ernst[1] and Aikaterini Mitrokotsa[1]

[1]University of St. Gallen {johannes.ernst, katerina.mitrokotsa}@unisg.ch

Abstract.

Despite its popularity, password based authentication is susceptible to various kinds of attacks, such as online or offline dictionary attacks. Employing biometric credentials in the authentication process can strengthen the provided security guarantees, but raises significant privacy concerns. This is mainly due to the inherent variability of biometric readings that prevents us from simply applying a standard hash function to them. In this paper we first propose an ideal functionality for modeling secure, privacy preserving biometric based two-factor authentication in the framework of universal composability (UC). The functionality is of independent interest and can be used to analyze other two-factor authentication protocols. We then present a generic protocol for biometric based two-factor authentication and prove its security (relative to our proposed functionality) in the UC framework. The first factor in our protocol is the possession of a device that stores the required secret keys and the second factor is the user's biometric template. Our construction can be instantiated with function hiding functional encryption, which computes for example the distance of the encrypted templates or the predicate indicating whether the templates are close enough. Our contribution can be split into three parts:

- We model privacy preserving biometric based two-factor authentication as an ideal functionality in the UC framework. To the best of our knowledge, this is the first description of an ideal functionality for biometric based two-factor authentication in the UC framework.
- We propose a general protocol that uses functional encryption and prove that it UC-realizes our ideal functionality.
- We show how to instantiate our framework with efficient, state of the art inner-product functional encryption. This allows the computation of the Euclidean distance, Hamming distance or cosine similarity between encrypted biometric templates. In order to show its practicality, we implemented our protocol and evaluated its performance.

Keywords: cryptographic protocols · biometric authentication · privacy preserving computation · universal composability

1 Introduction

It is long known that password based authentication is not very secure. Users tend to choose bad passwords and frequent leaks of password databases enable offline attacks. Biometric authentication mitigates many of these problems

and has better usability, since users do not have to remember their passwords anymore. However, the use of biometric authentication entails new problems. Biometric readings present inherent variability and, thus, common protection mechanisms such as standard hash functions or encryption cannot be employed. Furthermore, the client's biometric template (e.g. face, iris-scan or fingerprint) can be considered privacy sensitive, as it may reveal ethnic origin or even health conditions and, therefore, should remain private. Also, unlike passwords, biometrics cannot be changed, hence, the breach of a biometric database may have more severe consequences than the breach of a password database. Thus, for both privacy and security reasons it is important to achieve reliable authentication, while preventing the server from seeing the client's biometric templates. A general technique to make authentication more secure is two-factor authentication. This incorporates a second factor in the authentication process, so that in addition to the password or biometric the user needs for example a secret key that is stored on a phone or a secure hardware device. This makes an attack more difficult because it requires the attacker to get both the device and the biometric/password.

Homomorphic encryption and general multiparty computation have been used in the past to construct privacy preserving biometric authentication systems. However, using general multiparty computation usually requires multiple rounds of communication, which can reduce efficiency. Approaches using homomorphic encryption (e.g. [19]) have the problem that some part of the server needs to know the secret key to be able to decrypt the authentication decision. This also means that, when the entire server infrastructure is compromised, the attacker learns both the secret key and the encrypted templates and thereby the cleartext templates. Functional encryption (FE) is similar to homomorphic encryption in that it allows computing on encrypted data. The important advantage of FE in the biometric authentication setting is that it allows the server to learn the result of the computation in cleartext without having access to the secret key that allows decryption of the templates. Depending on the underlying FE scheme, the server only learns the distance of the biometric templates, or only the fact whether this distance is below a certain threshold. This makes FE well suited for the use case of biometric authentication.

Function hiding inner-product functional encryption (fh-IPFE) essentially allows computing the inner-product of two encrypted vectors. Therefore, fh-IPFE and its restricted and more efficient variant of single-key fh-IPFE has been used in the past to construct privacy preserving biometric authentication and identification systems ([7, 18, 15, 5]). Since one of the biometric templates is encoded as the function, the function hiding property is important, as otherwise this template would not be hidden. Because fh-IPFE only exists in the secret key setting and this key needs to be stored somewhere, it is very natural to consider fh-IPFE in the setting of two-factor authentication, where the first factor is the secret key stored on the device and the second factor is the user's biometric.

However, the security of these constructions has not been studied in the context of more complex systems in which multiple users have accounts on multiple

servers and the authentication protocol is executed in conjunction with other protocols. For example a user could try to authenticate towards two different servers at the same time with very similar or identical biometric templates. Even if both servers are corrupted they should learn no more than the output of the authentication protocol. It is also important that an attacker, who is in control of the network, can neither learn anything about the biometric templates, nor impersonate a legitimate user. Furthermore, a malicious server should not be able to use the authentication message of a user to impersonate that user to another server.

Let us illustrate the danger of too restrictive security models with a concrete example. For their biometric authentication system the authors of [7] only consider the privacy of the biometric templates in a setting with a single client and a single server. However, the model does not guarantee security, i.e. it does not guarantee that it is hard to impersonate an honest client. Concretely, in the security definition the adversary gets access to one oracle for each the enrolment phase and the authentication phase. In the real world these oracles call the actual IPFE scheme, whereas in the ideal world a simulator has to produce the answers without knowing the biometric template. This guarantees that an attacker cannot learn anything about the biometric templates from the ciphertexts. However, this does not guarantee that it is hard to impersonate an honest client. In fact, an attacker can send a random vector as ciphertext, which causes the output to be a random number. With their first parameter-set this is a random number in $\mathbb{Z}_{2^{20}}$. When the threshold for the biometric authentication system is e.g. $\tau = 32$, then the probability that a random number is below τ is 2^{-15}. This is a too high chance to impersonate a legitimate user for an attacker who does neither know the secret key nor the user's biometric. This example illustrates why it is important to model both privacy *and* security in complex environments.

Contribution: In order to fill this gap, we model biometric authentication as an ideal functionality in the framework of universal composability (UC) ([6]). This guarantees that the resulting protocols remain secure even in complex systems and in the presence of powerful attackers. Our ideal functionality can also serve as basis for the analysis of new protocols for two-factor authentication. In the second step we propose a biometric based two-factor authentication protocol that generically uses (single-key) function hiding functional encryption (fh-FE) and signatures and formally prove that it realizes our ideal functionality. The first factor is the secret key that is stored in secure hardware and the second factor is the biometric template. Next, we show how to instantiate our framework with (single-key) fh-IPFE schemes so that it can compute either the Euclidean distance, Hamming distance, or cosine similarity on encrypted biometric templates. We stress that our framework can also be instantiated in a way (e.g. with techniques of [5]) that does not leak the distance of the templates to the server, but only outputs the yes-no authentication decision. Finally, we describe the proof of concept implementation of our protocol and present the results of the performance tests. Thereby, we get a secure, privacy preserving and composable biometric authentication protocol. Furthermore, the protocol is practically effi-

cient and both enrolment and authentication only require a single message from the client to the server. Our contribution can be summarized as follows:

- We provide a UC formalization of (two-factor) biometric authentication.
- We propose a general protocol/framework that realizes our ideal functionality, which can be instantiated with fh-FE and signatures.
- We provide a concrete instantiation with fh-IPFE for computing the Euclidean distance, Hamming distance or cosine similarity and a proof of concept implementation.

1.1 Related Work

Universally Composable Biometric Authentication: Universally composable biometric authentication has received some attention in the literature ([11],[10],[1],[2]). Dupont et al. [10] considers fuzzy (symmetric) password authenticated key exchange, where the server also has to know the password in the clear. This problem is solved in [11] by moving to the asymmetric setting. Their protocol can be used for biometric authentication with binary vectors and the Hamming distance. Asymmetric password authenticated key exchange is a stronger notion of authentication than ours, since it considers mutual authentication, whereas in our case only the client authenticates to the server. However, it is usually less efficient, as Erwig et al. [11] note that "... going beyond password sizes of, say, 40 bits does not seem feasible.", where e.g. iris templates for the Hamming distance are usually more than 1000 bits long (see e.g. [8], [4]). Agrawal et al. [1] construct a biometric authentication system in which the reference template is secret shared among three client devices. However, having three connected devices whenever one wants to authenticate is not always practical. Agrawal et al. [2] consider the scenario in which a client wants to get access to a certain area and the client's device, an external terminal and service provider interact in order to perform the enrolment and the biometric matching. This is a different setting than ours.

Function Hiding Inner-Product Functional Encryption for Biometrics: Function hiding IPFE has already been used for biometric authentication ([7]) and identification ([18], [15], [5]), however their security models do not take composability into account. Composability is an important property when a protocols runs in conjunction with other protocols in a complex environment like the internet. Both [18] and [15] only rely on the security definition of IPFE but have no authentication or identification related security model. The security model of [5] does take biometrics into account, but is specific to searchable encryption. The work most closely related to ours is [7], because it is the only one that directly considers biometric *authentication*. Their security model is for biometric authentication, however it only considers one user and one server and does not guarantee that it is hard to impersonate another user. Both [7] and [18] have proposed constructions for fh-IPFE. Contrary to existing work, we propose for the first time a general protocol for privacy-preserving biometric based two-factor

authentication that provides UC security guarantees and can be instantiated using any suitable fh-FE scheme including the ones proposed by [7] and [18].

Other Biometric Authentication Protocols: There are many other protocols for privacy preserving biometric authentication, such as [24, 14, 25, 3, 13]. The authors of [24] and [13] consider biometric authentication for secure messaging and text search, respectively, which is a different setting than ours. The authors of [14], [25] and [3] consider the classical setting of a client authenticating to a server. The disadvantage of [3] and [14] is that the protocols need two non-colluding servers and that they are not secure against fully malicious attackers. The authors of [25], only consider the privacy of the client's biometrics, but not security against e.g. impersonations. None of them takes security under general composition into account.

Jarecki et al. [17] propose protocols for secure, password based two-factor authentication, where the user has a password and additionally owns a "crypto-capable device". They provide a very detailed security analysis of their protocol in a game base setting. The main difference to our setting is that we use the UC framework and that we work with biometrics instead of passwords.

2 Preliminaries

Notation: We write $[n]$ for $\{1, \ldots, n\}$. We use boldface letters such as $\mathbf{v}$ for vectors and we write v_i for the i-th entry of $\mathbf{v}$. With $\|\mathbf{v}\|$ we denote the L2 norm of $\mathbf{v}$. We write $\mathcal{C}$ for a client, $\mathcal{S}$ for a server, $\mathcal{A}$ for the adversary, Sim for the simulator and $\mathcal{Z}$ for the environment. With *reference* template we mean the biometric template used for enrolment and with *probe* template we mean the template used for authentication. We usually denote them as $\mathfrak{b}$ and $\mathfrak{b}'$.

2.1 (Secret Key) Function Hiding Functional Encryption

Definition 1. *A secret key functional encryption scheme for a set of functions $\mathcal{F}$, which map from $\mathcal{X}$ to $\mathcal{Y}$ consists of the following four algorithms:*

- $\mathsf{FE.Setup}(1^\lambda)$ *outputs public parameters* pp *and the master secret key* msk
- $\mathsf{FE.KeyGen}(\mathsf{msk}, f \in \mathcal{F})$ *outputs a functional decryption key* sk_f
- $\mathsf{FE.Enc}(\mathsf{msk}, x \in \mathcal{X})$ *outputs a ciphertext* c_x
- $\mathsf{FE.Dec}(\mathsf{sk}_f, c_x)$ *outputs a value* $y \in \mathcal{Y}$

We assume that pp *is given as implicit input to all other FE algorithms.*
Correctness: A functional encryption scheme FE *is correct if* $\forall \mathsf{msk} \leftarrow \mathsf{FE.Setup}(1^\lambda)$, $\forall f \in \mathcal{F}, \forall x \in \mathcal{X}$ *it holds that* $\mathsf{FE.Dec}(\mathsf{FE.KeyGen}(\mathsf{msk}, f), \mathsf{FE.Enc}(\mathsf{msk}, x)) = f(x)$.

In the special case of inner-product functional encryption (IPFE), which we use for our instantiation in Section 5, the inputs are $\mathcal{X} \coloneqq \mathbb{Z}_q^n \setminus \{0^n\}$ and $\mathcal{F} \coloneqq \{f_y : y \in \mathbb{Z}_q^n \setminus \{0^n\}\}$, where $f_y(x) \coloneqq \langle y, x \rangle$.

Definition 2. *For $b \in \{0,1\}$ we define $\mathsf{Exp}^b_{\mathcal{A}}(\lambda)$ as the following experiment: The experiment generates $(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{FE.Setup}(\lambda)$ and gives pp to $\mathcal{A}$. The experiment then answers the following types of oracle queries from $\mathcal{A}$:*

- $\mathsf{QKeyGen}(f_0, f_1)$*: The experiment returns* $\mathsf{sk} \leftarrow \mathsf{FE.KeyGen}(\mathsf{msk}, f_b)$.
- $\mathsf{QEnc}(x_0, x_1)$*: The experiment returns* $c \leftarrow \mathsf{FE.Enc}(\mathsf{msk}, x_b)$.

When $\mathcal{A}$ outputs a guess bit b', $\mathsf{Exp}^b_{\mathcal{A}}$ outputs the same b'.

Definition 3. *(Admissible adversaries) For an adversary $\mathcal{A}$, let $(f_0^1, f_1^1), \ldots, (f_0^{Q_K}, f_1^{Q_K})$ and $(x_0^1, x_1^1), \ldots, (x_0^{Q_E}, x_1^{Q_E})$ be the $\mathsf{QKeyGen}$ and QEnc queries. We say that an adversary is admissible if $\forall i \in [Q_K], \forall j \in [Q_E]\colon f_0^i(x_0^j) = f_1^i(x_1^j)$.*

Considering only admissible adversaries prevents $\mathcal{A}$ from trivially winning the game by querying functions and inputs with different output values. We say that an FE-scheme FE is fh-IND-secure, if for all admissible PPT adversaries $\mathcal{A}$ there is a negligible function $\mathsf{negl}(\lambda)$ such that for all large enough λ

$$|\Pr\left[\mathsf{Exp}^0_{\mathcal{A}}(\lambda) = 1\right] - \Pr\left[\mathsf{Exp}^1_{\mathcal{A}}(\lambda) = 1\right]| \leq \mathsf{negl}(\lambda)\,.$$

We say that an FE-scheme FE is *single-key* fh-IND-secure if the above holds for the modified version of the experiment were only the first $\mathsf{QKeyGen}$ query is answered, i.e. $\mathcal{A}$ only gets a single key. The function hiding property of the definition is given by the fact that $\mathcal{A}$ does not know whether they got a secret key for f_0 or f_1.

3 Modeling Biometric Authentication in the UC Framework

In this section we describe our approach to modeling biometric based two-factor authentication in the UC framework. We start by describing the threat model and proceed by motivating and stating our ideal functionality. We then explain the meaning of the functionality's different interfaces. Furthermore, we describe how we model message transfer and corruptions in the real world. We conclude by discussing some design decisions and the security and privacy guarantees that our functionality provides.

3.1 Threat Model

In this section we describe the threat model we consider. We will discuss the attacker's capabilities in more detail in Section 3.2, where we explain the different interfaces of the ideal functionality. We expect our protocol to be used for authentication over the internet. Communication is supposed to be done via TLS, which means that clients can send messages to a server and be sure that only the server can read them. On the other hand, the server does not know who the sender of a message is. We assume that the attacker controls the network. So the attacker notices when a message is sent. Although the attacker cannot

read the messages sent via TLS, they can delay or block them or inject their own messages.

The attacker can corrupt both arbitrary clients and the server and then control their actions in arbitrary ways, i.e. we consider malicious corruptions. For simplicity we only consider static corruptions of the server which means, that the server is corrupted either the entire time or not corrupted at all. Clients can be corrupted adaptively, i.e. the attacker can decide at any point in time to corrupt an arbitrary client. However, looking ahead, we will assume that the clients' keys are stored in secure hardware. So when the attacker corrupts a client, they will only gain control over that client's actions and gain blackbox access to the secure hardware. This means, they can give to the secure hardware instructions to enrol or authenticate, but they do not learn the internal state of the secure hardware. The use of secure hardware is also the reason why the attacker does not immediately learn the user's biometric reference template, even if they corrupt both the server and the client. Nevertheless, in the case that the adversary corrupts both the client *and* the server, they may be able to extract the biometric reference template (cf. Section 3.2 for an explanation of when this is possible).

3.2 The Ideal Functionality

We will first describe a simple ideal functionality in Figure 1 for two-factor authentication with biometrics as a second factor. Then we explain the changes that we applied to this simple functionality in order to get to the actual two-factor authentication functionality $\mathcal{F}_{\text{2FA}}^{\mathsf{out}}$ in Figure 2. Both functionalities are parameterized with the $\mathsf{out}(\cdot,\cdot)$ function, which will be part of our framework. It models both the output that the server gets from the protocol and the information that is leaked about the biometric templates. For example the out function could return the distance of the biometric templates, which would then mean that the server learns this distance. Alternatively the out function could only return one bit, indicating whether the biometric templates are close enough.

- On $(\textsc{Enrol}, \mathsf{sid}, \mathsf{uid}, \mathfrak{b})$ from $\mathcal{C}$
  - `if` there is *no* record $\langle \cdot, \cdot, \mathcal{C} \rangle$:
    - store $\langle \mathsf{uid}, \mathfrak{b}, \mathcal{C} \rangle$ and send $(\textsc{enrol}, \mathsf{sid}, \mathsf{uid})$ to $\mathcal{S}$
- On $(\textsc{Auth}, \mathsf{sid}, \mathsf{uid}, \mathfrak{b}')$ from $\mathcal{C}$:
  - `if` there is a record $\langle \mathsf{uid}, \mathfrak{b}, \mathcal{C} \rangle$:
    - send $(\textsc{auth}, \mathsf{sid}, \mathsf{uid}, \mathsf{out}(\mathfrak{b}, \mathfrak{b}'))$ to $\mathcal{S}$
  - `else` send $(\textsc{auth-fail}, \mathsf{sid})$ to $\mathcal{S}$

Fig. 1: The code of the simplified ideal functionality.

We want to capture the setting of two-factor authentication where the first factor is a secret key stored on the user's phone or a dedicated hardware device. The second factor is the user's biometric. In our model the client $\mathcal{C}$ is the device

on which the secret key is stored and the user is the actual person. For enrolment the client sends the user-id uid, such as an email address, and the biometric reference template $\mathfrak{b}$ to the ideal functionality. If that client is not yet enroled, the ideal functionality stores an internal record and notifies the server that a client with uid has enroled. Note that the server gets no information about $\mathfrak{b}$.

For the authentication, $\mathcal{C}$ again sends the user-id and a fresh biometric template $\mathfrak{b}'$ to the ideal functionality. If and only if the same client has previously enroled with the same user-id, the ideal functionality gives $\mathsf{out}(\mathfrak{b}, \mathfrak{b}')$ to $\mathcal{S}$. Depending on the underlying FE scheme this can be the distance of the templates or the yes-no authentication decision. Note that apart from $\mathsf{out}(\mathfrak{b}, \mathfrak{b}')$ the server learns nothing about the biometric templates. The fact that the ideal functionality checks that the identity $\mathcal{C}$ of the client in the authentication phase is the same as the identity of the client in the enrolment phase represents the first factor, i.e. possession of the secret key. Because in the UC framework clients cannot fake their identity, a client cannot authenticate with the uid of another client. The second factor, i.e. the biometric authentication is represented by the ideal functionality storing the biometric templates and giving $\mathsf{out}(\mathfrak{b}, \mathfrak{b}')$ to the server.

This simple functionality, however, does not capture all actions that an adversary can take in the real world. Therefore, we added several interfaces in order to model the adversary's capabilities. Also we exchanged the uid by a randomly chosen rid to avoid that two clients enrol with the same identifier. The resulting functionality $\mathcal{F}_{\mathrm{2FA}}^{\mathsf{out}}$ is shown in Figure 2.

Meaning of the interfaces of $\mathcal{F}_{2FA}^{\mathsf{out}}$: Here we explain the meaning of all the interfaces of $\mathcal{F}_{\mathrm{2FA}}^{\mathsf{out}}$ and their connection to reality.

The $(\textsc{Enrol}, \mathsf{sid}, \mathsf{ssid}, \mathfrak{b})$ and $(\textsc{Auth}, \mathsf{sid}, \mathsf{ssid}, \mathfrak{b})$ interface of a client $\mathcal{C}$: These are the interfaces that a client would use for normal enrolment and authentication. All other interfaces are there to model the adversary's capabilities. We let the environment $\mathcal{Z}$ choose the biometric template and the time of enrolment/authentication, because in practice we do not have control over the biometric template or the timing. In UC-PAKE for example the environment also chooses the credential i.e. the password and the time of enrolment/authentication (cf. [16]). Letting the environment choose the biometric templates of the clients and the time (and order) of their enrolments/authentications is in a sense like taking the worst case of possible events in reality.

The EnrolOK and AuthOK interfaces: In the real world the adversary can delay or block messages. The EnrolOK and AuthOK interfaces model the fact that messages are only delivered when $\mathcal{A}$ explicitly allows this.

The $(\textsc{Enrol}, \mathsf{sid}, \mathsf{ssid}, \mathsf{rid}, \mathfrak{b})$ and $(\textsc{Auth}, \mathsf{sid}, \mathsf{ssid}, \mathsf{rid}, \mathfrak{b}')$ interface of $\mathcal{A}$: These two interfaces are necessary because the adversary has more control over the records of corrupted clients. Our ideal functionality in Figure 2 has an Enrol interface which explicitly allows the adversary to enrol malicious clients with arbitrary rid. All clients enroled in that way are explicitly stored as *adversarially enroled* clients by $\mathcal{F}_{\mathrm{2FA}}^{\mathsf{out}}$. The adversary can then use its Auth interface to authenticate in the name of any such client. Importantly, the adversary has, through their Enrol and Auth interfaces, only influence on the records of the

This functionality interacts with a server $\mathcal{S}$ (specified in the sid) and arbitrary clients. It is parameterized by the function $\mathsf{out}(\cdot,\cdot)$, which determines the output that $\mathcal{S}$ gets.

Enrolment:

- On $(\textsc{Enrol}, \mathsf{sid}, \mathsf{ssid}, \mathfrak{b})$ from $\mathcal{C}$
  - `if` there is *no* record $\langle \text{enrol-request}, \cdot, \cdot, \mathcal{C}\rangle$:
    - store $\langle \text{enrol-request}, \mathsf{ssid}, \mathfrak{b}, \mathcal{C}\rangle$
    - send $(\textsc{enrol}, \mathsf{sid}, \mathsf{ssid}, \mathcal{C})$ to $\mathcal{A}$
- On $(\textsc{EnrolOK}, \mathsf{sid}, \mathsf{ssid})$ from $\mathcal{A}$
  - `if` there is a record $\langle \text{enrol-request}, \mathsf{ssid}, \mathfrak{b}, \mathcal{C}\rangle$ and *no* record $\langle \text{enroled}, \mathcal{C}, \cdot, \cdot\rangle$:
    - sample $\mathsf{rid} \leftarrow_\$ \{0,1\}^\lambda$
    - store $\langle \text{enroled}, \mathcal{C}, \mathsf{rid}, \mathfrak{b}\rangle$
    - `if` the record $\langle \text{enrol-request}, \mathsf{ssid}, \mathfrak{b}, \mathcal{C}\rangle$ is marked CORRUPTED:
      - mark the record $\langle \text{enroled}, \mathcal{C}, \mathsf{rid}, \mathfrak{b}\rangle$ as CORRUPTED
    - send $(\textsc{enrol}, \mathsf{sid}, \mathsf{ssid}, \mathsf{rid})$ to $\mathcal{S}$
- On $(\textsc{Enrol}, \mathsf{sid}, \mathsf{ssid}, \mathsf{rid}, \mathfrak{b})$ from $\mathcal{A}$:
  - `if` there is a record $\langle \text{enroled-adversarial}, \mathsf{rid}, \cdot\rangle$ or $\langle \text{enroled}, \cdot, \mathsf{rid}, \cdot\rangle$:
    - send $(\textsc{enrol}, \mathsf{sid}, \mathsf{ssid}, \perp)$ to $\mathcal{S}$
  - `else`: store $\langle \text{enroled-adversarial}, \mathsf{rid}, \mathfrak{b}\rangle$ and send $(\textsc{enrol}, \mathsf{sid}, \mathsf{ssid}, \mathsf{rid})$ to $\mathcal{S}$

Authentication:

- On $(\textsc{Auth}, \mathsf{sid}, \mathsf{ssid}, \mathfrak{b}')$ from $\mathcal{C}$:
  - `if` there is a record $\langle \text{enrol-request}, \cdot, \cdot, \mathcal{C}\rangle$:
    - store $\langle \text{auth-request}, \mathsf{ssid}, \mathfrak{b}', \mathcal{C}\rangle$
    - send $(\textsc{auth}, \mathsf{sid}, \mathsf{ssid}, \mathcal{C})$ to $\mathcal{A}$
- On $(\textsc{AuthOK}, \mathsf{sid}, \mathsf{ssid})$ from $\mathcal{A}$:
  - `if` there is a record $\langle \text{auth-request}, \mathsf{ssid}, \mathfrak{b}', \mathcal{C}\rangle$:
    - `if` there is a record $\langle \text{enroled}, \mathcal{C}, \mathsf{rid}, \mathfrak{b}\rangle$:
      - delete the record $\langle \text{auth-request}, \mathsf{ssid}, \mathfrak{b}', \mathcal{C}\rangle$ and send $(\textsc{auth}, \mathsf{sid}, \mathsf{ssid}, \mathsf{rid}, \mathsf{out}(\mathfrak{b}, \mathfrak{b}'))$ to $\mathcal{S}$
    - `else` delete the record $\langle \text{auth-request}, \mathsf{ssid}, \mathfrak{b}', \mathcal{C}\rangle$ and send $(\textsc{auth-fail}, \mathsf{sid}, \mathsf{ssid})$ to $\mathcal{S}$
- On $(\textsc{Auth}, \mathsf{sid}, \mathsf{ssid}, \mathsf{rid}, \mathfrak{b}')$ from $\mathcal{A}$:
  - `if` there is a record $\langle \text{enroled-adversarial}, \mathsf{rid}, \mathfrak{b}\rangle$
    - send $(\textsc{auth}, \mathsf{sid}, \mathsf{ssid}, \mathsf{rid}, \mathsf{out}(\mathfrak{b}, \mathfrak{b}'))$ to $\mathcal{S}$
  - `else` send $(\textsc{auth-fail}, \mathsf{sid}, \mathsf{ssid})$ to $\mathcal{S}$

Corruption and impersonation:

- On $(\textsc{Corrupt}, \mathsf{sid}, \mathcal{C})$ from $\mathcal{A}$:
  - `if` there is a record $\langle \text{enroled}, \mathcal{C}, \mathsf{rid}, \mathfrak{b}\rangle$:
    - mark the record CORRUPTED
    - send $(\textsc{corrupted}, \mathsf{sid}, \mathsf{rid})$ to $\mathcal{A}$
  - `if` there is a record $\langle \text{enrol-request}, \mathsf{ssid}, \cdot, \mathcal{C}\rangle$:
    - mark the record CORRUPTED
    - send $(\textsc{corrupted}, \mathsf{sid}, \mathsf{rid})$ to $\mathcal{A}$
- On $(\textsc{TryImpersonate}, \mathsf{sid}, \mathsf{ssid}, \mathcal{C}, \mathfrak{b}')$ from $\mathcal{A}$:
  - `if` there is a record $\langle \text{enroled}, \mathcal{C}, \mathsf{rid}, \mathfrak{b}\rangle$ that is marked corrupted:
    - send $(\textsc{auth}, \mathsf{sid}, \mathsf{ssid}, \mathsf{rid}, \mathsf{out}(\mathfrak{b}, \mathfrak{b}'))$ to $\mathcal{S}$

Fig. 2: The code of the ideal functionality $\mathcal{F}_{\text{2FA}}^{\mathsf{out}}$.

adversarially enroled clients. This guarantees that the adversary cannot authenticate in the name of an honest client and, thus, cannot impersonate an honest client by using this interface.

The CORRUPT and TRYIMPERSONATE interfaces: The adversary can use the CORRUPT interface to corrupt an honest client and thereby bypass the first authentication factor. In reality this could mean that the adversary has gained remote access to the device that the client uses. It can also mean that the adversary has stolen the device or secure hardware token and has physical access to it. These two cases are modeled by the same interface, because in both cases the adversary gains the same capabilities, namely to try to impersonate the client with a self-chosen biometric template. This is modeled by the TRYIMPERSONATE interface, where $\mathcal{A}$ can specify a client $\mathcal{C}$ and a biometric template $\mathfrak{b}'$. If $\mathcal{C}$ has been corrupted previously, $\mathcal{F}_{\text{2FA}}^{\mathsf{out}}$ will send $(\text{AUTH}, \mathsf{sid}, \mathsf{ssid}, \mathsf{rid}, \mathsf{out}(\mathfrak{b}, \mathfrak{b}'))$ to the server $\mathcal{S}$. Thus, essentially the impersonation attempt will only succeed if $\mathcal{A}$ knows a biometric template $\mathfrak{b}'$ that is close enough to the reference template $\mathfrak{b}$ and thereby bypasses the second authentication factor. This meaningfully models reality where the adversary can only successfully impersonate an honest client if they both have access to the device and know a matching biometric template.

Modeling corruptions in the real world: The code in Figure 3 models the client's behavior upon corruption. This code is not part of the protocol that would be executed in reality, but is added to the client in the real world experiment in the UC framework in order to properly model client corruptions. In the language of the UC framework, this code describes the behavior of the client's *shell*, whereas the *body* contains the actual protocol code. Later in our protocol we assume the use of secure hardware for storing key material and computing the enrolment and authentication messages, which is also reflected in the code in Figure 3. Namely, the adversary does not get the internal state of the client, but only blackbox access to the secure hardware. This means that $\mathcal{A}$ can try to impersonate a client by giving a biometric template $\mathfrak{b}'$ to the secure hardware and forwarding the message to the server. We distinguish the *host* from the secure hardware. The host would usually be the software on a smartphone or laptop and the secure hardware would be a special chip in the phone or laptop, or a USB-stick like hardware token.

When one wants to instantiate our model without the use of secure hardware then it is necessary to adapt the code in Figure 3 to return the entire local state of the client and then exactly follow the instructions from $\mathcal{A}$.

Modeling message transfer in the real world: Messages in our framework will be sent via the $\mathcal{F}'_{\text{SMT}}$ functionality (Figure 4), which is an adaption of $\mathcal{F}_{\text{SMT}}$ from [6]. It is meant to model TLS connections where the server is authenticated via a certificate but the client is not authenticated. This is modeled by the fact that $\mathcal{F}'_{\text{SMT}}$, as opposed to $\mathcal{F}_{\text{SMT}}$, does not give the client's identity $\mathcal{C}$ to the server. However, the client can be sure that only the server can read the message. It is important to model this, because giving the client's identity $\mathcal{C}$ to the server

This code describes the behavior (of the shell) of a client $\mathcal{C}$ upon corruption.

- On $(\textsc{Corrupt}, \mathsf{sid})$ from $\mathcal{A}$
  - `if` $\mathcal{C}$ is enroled (i.e. $(\textsc{Enrol}, \cdot, \cdot)$) has been called before:
    - set flag corrupted
    - from now on ignore every input from $\mathcal{Z}$ and only take instructions from $\mathcal{A}$
    - send $(\textsc{corrupted}, \mathsf{sid})$ to $\mathcal{A}$
- On $(\textsc{TryImpersonate}, \mathsf{sid}, \mathsf{ssid}, \mathfrak{b}')$ from $\mathcal{A}$
  - `if` flag corrupted is set
    - give $(\textsc{Auth}, \mathsf{sid}, \mathsf{ssid}, \mathfrak{b}')$ to the secure hardware in the name of the host
    - when getting an output m from the secure hardware, send m to $\mathcal{A}$

Fig. 3: The code modeling client behavior corruption.

- On $(\textsc{Send}, \mathsf{sid}, \mathcal{S}, m)$ from a client $\mathcal{C}$
  - send $(\textsc{sent}, \mathsf{sid}, \mathcal{C}, \mathcal{S}, l(m))$ to $\mathcal{A}$
- On $(\textsc{ok}, \mathsf{sid})$ from $\mathcal{A}$:
  - If not yet generated output then send $(\textsc{Sent}, \mathsf{sid}, m)$ to server $\mathcal{S}$

Fig. 4: The code of the $\mathcal{F}'_{\text{SMT}}$ functionality (adapted from [6] to mimic TLS messages).

would make further authentication unnecessary. Simply put, we will use $\mathcal{F}'_{\text{SMT}}$ to capture the client's instruction "send m to $\mathcal{S}$ over the internet via TLS".

On using rid *instead of* uid*:* It is important that the server gets a unique identifier with which they can link account information such as the user's bank account. We chose to replace the environment supplied uid by a rid that is randomly chosen by $\mathcal{F}_{\text{2FA}}^{\mathsf{out}}$ in the enrolment phase. This reduces the options of $\mathcal{Z}$, as now $\mathcal{Z}$ cannot make two honest clients enrol with the same identifier. Of course the server can still store the user's email address alongside to the user record.

On (not) including an interface for the adversary to guess the biometric: Other authentication functionalities such as in [16] and [11] have an interface that $\mathcal{A}$ can use to make online or offline password guesses. Our functionality does not have an interfaces for offline guesses of credentials, because even if the server is corrupted, our protocol protects against such an attack. This is possible, because as opposed to [16] and [11] in our case the client stores secret key material. Looking ahead, the only way that $\mathcal{A}$ could perform an offline attack is after both corrupting the server and the client *and* breaking the security of the client's secure hardware module. Note that it is not enough to corrupt the client and break the security of the hardware module, because all data, which the hardware module stores, are independent of the biometrics. Regardless, our protocol builds on the assumption that the secure hardware is indeed secure and, thus, such an attack is out of scope of our model.

The TryImpersonate-interface can be seen as a way of allowing $\mathcal{A}$ to perform online credential guesses for specific clients. Note, however, that this is only possible after $\mathcal{A}$ has corrupted the respective client.

On using passwords instead of biometrics: Although our ideal functionality in Figure 2 is aimed at capturing biometric authentication, it does in fact also capture password based authentication. Whether the inputs to the interfaces for the functionality are biometric templates or passwords does not matter. In order to perform equality checks on the passwords, one simply has to define the $\mathsf{out}(\cdot,\cdot)$ function to return 1 if both arguments are equal (i.e. the passwords match), and 0 otherwise. Additionally our functionality naturally captures typo tolerance in the password matching. For that, the $\mathsf{out}(\cdot,\cdot)$ function would be set to return 1 if the passwords match except for some common typos. This increases the applicability of our functionality to a wider range of scenarios.

Discussion of the security guarantees: The functionality $\mathcal{F}_{\text{2FA}}^{\mathsf{out}}$ guarantees that an attacker cannot impersonate an honest client without corrupting that client *and* having a matching biometric template. The only way for $\mathcal{A}$ to impersonate a client is through the TryImpersonate interface, after calling the Corrupt interface.

The functionality also guarantees that nobody learns anything about the biometric templates, except of what can be inferred from the out function. This can be seen by observing that $\mathcal{F}_{\text{2FA}}^{\mathsf{out}}$'s behavior does not depend on the biometric templates and the only way a biometric template appears in $\mathcal{F}_{\text{2FA}}^{\mathsf{out}}$'s output is within $\mathsf{out}(\mathfrak{b},\mathfrak{b}')$. This holds even if the server is malicious and an arbitrary number of clients is corrupted. The same holds for the impossibility of offline guessing attacks, as $\mathcal{F}_{\text{2FA}}^{\mathsf{out}}$ simply does not allow them. However, if the adversary corrupts both a client *and* the server, then they can try to impersonate that client with arbitrary biometric templates and at the same time learn the result of the out-function. When the out-function is the distance of the two templates, then the adversary can often reconstruct the corrupted client's reference template from these results. Note however, that this is only possible when both the client and the server are corrupted. Under these circumstances many protocols do not retain any security guarantees. Our protocol in Section 4.3 achieves these strong guarantees by using secure hardware on the client side.

Although the ideal functionality is for a single server, the composition theorem of [6] guarantees security also in situations with multiple servers, when there is one instance of $\mathcal{F}_{\text{2FA}}^{\mathsf{out}}$ per server. A single user can also have multiple accounts at possibly different servers. In that case the user's device would run multiple instances of the protocol in Figure 6. Again the composition theorem of [6] ensures that the security guarantees of $\mathcal{F}_{\text{2FA}}^{\mathsf{out}}$ still hold.

4 The Framework

In this section we describe our general protocol Π_{2FA} and show that it UC realizes $\mathcal{F}_{\text{2FA}}^{\mathsf{out}}$. We start by describing a concrete use case in which our protocol could be used in practice. We then define several algorithms that are used in our protocol and the security proof and define their correctness. Every instantiation of our general protocol needs to instantiate these algorithms. Then we describe our general protocol and prove its security.

4.1 Use Case

In our use case in Figure 5 the user has a device such as a smartphone or laptop (the host in Figure 6) and wants to authenticate to a server. The secret keys of the FE scheme and the signature scheme are stored in secure hardware. This can be an external hardware token as in Figure 5, or a trusted execution environment on the phone such as ARM's TrustZone. For enrolment and authentication the device takes the user's biometric template and gives it to the secure hardware. The secure hardware can optionally do liveness detection, i.e. trying to detect if the biometric data is coming from a live person or if e.g. somebody is holding a photo in front of the camera. In the next step, the secure hardware encodes and encrypts the biometric template and sends the resulting message to the server. The server then performs the enrolment or authentication. Thus, the client can authenticate to the server in a secure and privacy preserving manner. The first factor of the authentication is the possession of the hardware token and the second factor is the user's biometric. When an attacker compromises the user's host device or steals the hardware token, they cannot impersonate the user, as they are lacking the user's biometric.

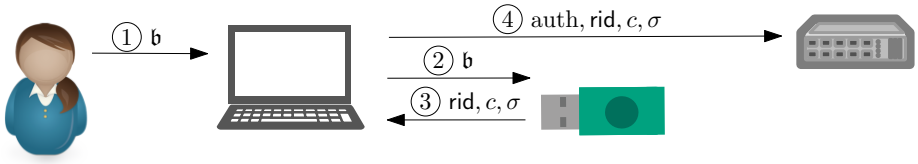

Fig. 5: The authentication phase of our use case where the secret keys are stored on a secure hardware token.

This use case is very similar to the setting of FIDO2 [12]. However, in FIDO2 the biometric matching is performed in the secure hardware. Both with our protocol in Section 4.3 and with FIDO2, if the attacker manages to compromise the keys in the secure hardware, then they can impersonate the user. However, an advantage of our protocol over FIDO2 is that in this case the user's biometric reference template remains secret, because the data stored on the secure hardware do not depend on the reference template.

4.2 Requirements

Let $\mathsf{Sig} = (\mathsf{Sig.Gen}, \mathsf{Sig.Sign}, \mathsf{Sig.Vfy})$ be a EUF-CMA secure signature scheme. Let $\mathsf{FE} = (\mathsf{FE.Setup}, \mathsf{FE.KeyGen}, \mathsf{FE.Enc}, \mathsf{FE.Dec})$ be a fh-IND secure FE scheme for the family of functions $\mathcal{F}$, the input space $\mathcal{X}$ and output space $\mathcal{Y}$. To instantiate our framework the following six algorithms have to be defined and fulfill certain properties:

- $\mathsf{encodeRef}(\cdot)\colon \mathfrak{B} \to \mathcal{F}$, takes the reference template and encodes it as function of the FE scheme

- $\mathsf{encodeProbe}(\cdot)\colon \mathfrak{B} \to \mathcal{X}$, takes the probe template and encodes it as input of the FE scheme
- $\mathsf{out}(\cdot,\cdot)\colon \mathfrak{B} \times \mathfrak{B} \to \mathcal{D}$, takes two biometric templates and outputs the authentication result
- $\mathsf{FE2out}(\cdot)\colon \mathcal{Y} \to \mathcal{D}$, takes an output of $\mathsf{FE.Dec}$ and converts it to an authentication result
- $\mathsf{chooseFakeRef}()$, outputs a fake reference template to be used by the simulator
- $\mathsf{chooseFakeProbe}(\cdot,\cdot)\colon \mathfrak{B} \times \mathcal{D} \to \mathfrak{B}$, takes a reference template and a desired result and outputs a corresponding fake probe template

We say that $(\mathsf{encodeRef}, \mathsf{encodeProbe}, \mathsf{out}, \mathsf{FE2out})$ are correct if $\forall \mathfrak{b}, \mathfrak{b}' \in \mathfrak{B}$, $\mathsf{msk} \leftarrow \mathsf{FE.Setup}(1^\lambda), \mathsf{sk}_{\mathfrak{b}} \leftarrow \mathsf{FE.KeyGen}(\mathsf{msk}, \mathsf{encodeRef}(\mathfrak{b})), c \leftarrow \mathsf{FE.Enc}(\mathsf{msk}, \mathsf{encodeProbe}(\mathfrak{b}'))$:

$$\mathsf{out}(\mathfrak{b}, \mathfrak{b}') = \mathsf{FE2out}(\mathsf{FE.Dec}(\mathsf{sk}_{\mathfrak{b}}, c)).$$

We say that $(\mathsf{chooseFakeRef}, \mathsf{chooseFakeProbe})$ are correct if $\forall d \in \mathcal{D}$, $\mathfrak{b} \leftarrow \mathsf{chooseFakeRef}(), \mathfrak{b}' \leftarrow \mathsf{chooseFakeProbe}(\mathfrak{b}, d)\colon d = \mathsf{out}(\mathfrak{b}, \mathfrak{b}')$. Choosing fake templates will later be necessary for the simulator and allows us to rely on the weaker fh-IND security instead of simulation based security of the FE scheme. We will instantiate these algorithms in Section 5.

4.3 The Protocol

The code of the client and the server of our protocol Π_{2FA} are depicted in Figure 6 and Figure 7. We divide the client in two parts, the *host* and the *secure hardware*. The host is the normal program on the laptop or the phone, whereas the code of the secure hardware runs in a trusted execution environment or on an external hardware token. We use $\mathcal{F}'_{\mathrm{SMT}}$ to model TLS messages sent by the client to the server as described in Figure 4 and Section 3.2. Note that in our version of $\mathcal{F}'_{\mathrm{SMT}}$ (as opposed to $\mathcal{F}_{\mathrm{SMT}}$ from [6]), the client's identity $\mathcal{C}$ is *not* given to the server.

For enrolling, the host simply forwards the instruction to the secure hardware, which chooses a random rid and keys for the signature scheme and FE scheme. The secure hardware then encodes the biometric reference template and generates a FE key for the encoded template. It gives the rid, the FE key and the signature public key to the host, which sends it to the server. The server simply stores these data. The secure hardware stores the FE master secret key, the signature secret key and rid.

For authenticating, the host again passes the instruction to the secure hardware, which first checks if it is already enroled. If so, it encodes the probe template and encrypts it with the FE scheme. Furthermore it signs $(\mathsf{sid}, \mathsf{ssid}, \mathsf{rid}, c)$ in order to prove ownership of the signature secret key. The secure hardware gives the rid, the FE ciphertext and the signature to the host, who sends the message to the server. The server checks the signature and computes the output of the protocol. We assume that ssid is unique per client i.e. per rid. Thus, the value

$(\mathsf{sid}, \mathsf{ssid}, \mathsf{rid})$ is globally unique and prevents an attacker from reusing the signature. In practice the sid can be set as the server's name and the ssid as a counter that is increased by the client at each authentication. The server would also keep a counter per client and only accept messages with a counter value higher than the stored one. Upon receiving the signature and the encrypted probe template, the server checks the signature and uses the $\mathsf{FE.Dec}$ algorithm to compute the output, which for example could be the distance of the biometric templates or a yes-no decision.

The code of the client, which is divided in the host and the secure hardware :

– the host on $(\textsc{Enrol}, \mathsf{sid}, \mathsf{ssid}, \mathfrak{b})$ from $\mathcal{Z}$:
- give $(\textsc{Enrol}, \mathsf{sid}, \mathsf{ssid}, \mathfrak{b})$ to the secure hardware

  – the secure hardware on getting $(\textsc{Enrol}, \mathsf{sid}, \mathsf{ssid}, \mathfrak{b})$ from the host
  - if there is *no* record $\langle \mathsf{rid}, (\cdot, \cdot), \cdot \rangle$:
    * $\mathsf{rid} \leftarrow\$ \{0,1\}^\lambda$
    * $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Sig.Gen}(1^\lambda)$
    * $\mathsf{msk} \leftarrow \mathsf{FE.Setup}(1^\lambda)$
    * $\mathbf{b} := \mathsf{encodeRef}(\mathfrak{b})$
    * $\mathsf{sk}_{\mathbf{b}} \leftarrow \mathsf{FE.KeyGen}(\mathsf{msk}, \mathbf{b})$
    * store record $\langle \mathsf{rid}, (\mathsf{pk}, \mathsf{sk}), \mathsf{msk} \rangle$
    * give back $m := (\mathrm{enrol}, \mathsf{rid}, \mathsf{pk}, \mathsf{sk}_{\mathbf{b}}))$ to the host

- on getting m from the secure hardware
  * send $(\textsc{Send}, (\mathsf{sid}, \mathsf{ssid}), \mathcal{S}, m)$ to $\mathcal{F}'_{\mathrm{SMT}}$ as message to $\mathcal{S}$

– the host on $(\textsc{Auth}, \mathsf{sid}, \mathsf{ssid}, \mathfrak{b}')$ from $\mathcal{Z}$:
- give $(\textsc{Auth}, \mathsf{sid}, \mathsf{ssid}, \mathfrak{b}')$ to the secure hardware

  – the secure hardware on getting $(\textsc{Auth}, \mathsf{sid}, \mathsf{ssid}, \mathfrak{b}')$ from the host
  - if there is a record $\langle \mathsf{rid}, (\mathsf{pk}, \mathsf{sk}), \mathsf{msk} \rangle$
    * $\mathbf{b}' := \mathsf{encodeProbe}(\mathfrak{b}')$
    * $c \leftarrow \mathsf{FE.Enc}(\mathsf{msk}, \mathbf{b}')$
    * $\sigma \leftarrow \mathsf{Sig.Sign}(\mathsf{sk}, (\mathsf{sid}, \mathsf{ssid}, \mathsf{rid}, c))$
    * give back $m := (\mathrm{auth}, \mathsf{rid}, c, \sigma))$ to the host

- on getting m from the secure hardware
  * send $(\textsc{Send}, (\mathsf{sid}, \mathsf{ssid}), \mathcal{S}, m)$ to $\mathcal{F}'_{\mathrm{SMT}}$ as message to $\mathcal{S}$

Fig. 6: The code of the client.

On the need for secure hardware: Without secure hardware, the adversary can —by compromising the client —get the secret keys and they could be able to choose a fake encoding of a biometric template, which may convince the server that the attacker is authentic. For example when we instantiate the FE scheme with an IPFE scheme, as we do in Section 5, then the attacker can choose $\mathbf{b}' = \big(0 \dots 0\big)^\top$. This makes the inner-product with the encoded reference template to be 0, which may convince the server that the distance is zero and allows the attacker to impersonate the user. This problem seems to be inherent to IPFE schemes and is also present in e.g [7] and [18]. We chose to address this

The code of the server:

- on $(\text{SENT}, (\mathsf{sid}, \mathsf{ssid}), m = (\text{enrol}, \mathsf{rid}, \mathsf{pk}, \mathsf{sk_b}))$ from $\mathcal{F}'_{\text{SMT}}$ as message from some client:
  - `if` there is no record $\langle \mathsf{rid}, \cdot, \cdot \rangle$:
    - store $\langle \mathsf{rid}, \mathsf{pk}, \mathsf{sk_b} \rangle$
    - output $(\text{ENROL}, \mathsf{sid}, \mathsf{ssid}, \mathsf{rid})$
  - `else` output $(\text{ENROL}, \mathsf{sid}, \mathsf{ssid}, \perp)$
- on $(\text{SENT}, (\mathsf{sid}, \mathsf{ssid}), m = (\text{auth}, \mathsf{rid}, c, \sigma))$ from $\mathcal{F}'_{\text{SMT}}$ as message from some client:
  - `if` there is a record $\langle \mathsf{rid}, \mathsf{pk}, \mathsf{sk_b} \rangle$ and $\mathsf{Sig.Vfy}(\mathsf{pk}, (\mathsf{sid}, \mathsf{ssid}, \mathsf{rid}, c), \sigma) = 1$:
    - $d := \mathsf{FE2out}(\mathsf{FE.Dec}(\mathsf{sk_b}, c))$
    - output $(\text{AUTH}, \mathsf{sid}, \mathsf{ssid}, \mathsf{rid}, d)$
  - `else` output $(\text{AUTH-FAIL}, \mathsf{sid}, \mathsf{ssid})$

Fig. 7: The code of the server.

problem by storing the client's secret keys in secure hardware and letting the secure hardware do the encoding of the biometric template. This ensures that only correctly encoded templates are encrypted and signed. Another approach to prevent this attack is to add zero knowledge proofs to the protocol. The client would then have to prove to the server that the biometric template has been correctly encoded before being encrypted.

4.4 Security Proof

Theorem 1. *If* FE *is a (single-key) fh-*IND *secure fh-FE scheme and* Sig *is an* EUF-CMA *secure signature scheme, then* Π_{2FA} *UC-emulates* $\mathcal{F}_{2FA}^{\mathsf{out}}$ *in the* $\mathcal{F}'_{SMT}$ *hybrid model in the presence malicious adversaries.*

We defer the security proof to Appendix A

5 Instantiation

Function hiding IPFE schemes allow computing inner-products between two encrypted vectors. This enables us to evaluate any distance measure between biometric templates that can be written as inner-product. When we then use fh-IPFE schemes in our framework, this enables the server to compute the distance between the client's reference and probe template. In this section we describe how to use an fh-IPFE scheme IPFE to instantiate our framework for computing the Euclidean distance, the Hamming distance or the cosine similarity of two biometric templates. We also sketch how the techniques of [5] can be used to instantiate our framework in a way that avoids the leakage of the distance of the biometric templates.

Our framework can be instantiated with both [7] and [18]. Both provide function hiding simulation security, which implies fh-IND security which is the notion that we need in our framework. The IPFE scheme of Cheon et al. [7] only allows the creation of a single decryption key, which is sufficient in our setting and allows the scheme to be quite efficient. The scheme relies on the security of the learning with errors assumption. The IPFE scheme of Kim et al. [18] allows

an arbitrary number of decryption keys, which makes it less efficient. It uses pairings and has a security proof in the generic group model. We use the IPFE scheme as black-box, the parties call the IPFE.Setup, IPFE.KeyGen, IPFE.Enc, IPFE.Dec algorithms whenever the respective algorithm in Figure 6 and Figure 7 is called. We will assume that our biometric templates $\mathfrak{b}$ are already embedded in e.g. Euclidean or Hamming space, i.e. they are vectors were a small distance implies similar biometrics. This is often achieved by applying a neural network to the biometric reading ([23, 9]). Letting $\mathcal{Z}$ directly choose the embedding can be seen as modeling $\mathcal{Z}$'s influence on the network creation and training. Next we show how to instantiate the algorithms described in Section 4.2.

Squared Euclidean Distance: In [23] the authors show how to transform face biometrics into vectors, where similar faces have low squared Euclidean distance. Let the discretized embedding be vectors in $\mathbb{Z}_{m+1}^n$ (in [23] $n = 128$). Then the maximum squared Euclidean distance is $m^2 \cdot n$ and we can set $\mathcal{D} := \{0, \ldots, m^2 \cdot n\}$. To later simplify the chooseFakeRef and chooseFakeProbe algorithms of the simulator we set $\mathfrak{B} := \mathbb{Z}_q^n$, for $q > m^2 \cdot n$. Note that this also allows $\mathcal{Z}$ to choose biometric templates with too large coordinates, however, this does not help in finding a vector that is close to the reference template and, thus, does not impact security. Because the desired output is the squared Euclidean distance d_E, we define $\mathsf{out}(\mathfrak{b}, \mathfrak{b}') := \min(d_E(\mathfrak{b}, \mathfrak{b}'), m^2 \cdot n)$. Note that the squared Euclidean distance between $\mathfrak{b}$ and $\mathfrak{b}'$ can also be written as $d_E(\mathfrak{b}, \mathfrak{b}') = -2\langle \mathfrak{b}, \mathfrak{b}' \rangle + \|\mathfrak{b}\|^2 + \|\mathfrak{b}'\|^2$. Thus, we can define $\mathsf{encodeRef}(\mathfrak{b}) := \mathbf{b} = \begin{pmatrix} \mathfrak{b}_1 & \ldots & \mathfrak{b}_n & 1 & \|\mathfrak{b}\|^2 \end{pmatrix}^\top$ and $\mathsf{encodeProbe}(\mathfrak{b}') := \mathbf{b}' = \begin{pmatrix} -2\mathfrak{b}'_1 & \ldots & -2\mathfrak{b}'_n & \|\mathfrak{b}'\|^2 & 1 \end{pmatrix}^\top$. Then $\langle \mathbf{b}, \mathbf{b}' \rangle$ is the squared Euclidean distance of $\mathfrak{b}$ and $\mathfrak{b}'$. This requires an IPFE scheme with $\mathcal{X} = \mathbb{Z}_q^{n+2}$ and $\mathcal{Y} = \mathbb{Z}_q$. Finally we define $\mathsf{FE2out}(d) := \min(d, m^2 \cdot n)$. Capping the result at $m^2 \cdot n$ is necessary to ensure that it stays inside of $\mathcal{D}$. Kim et al. [18] used the same encoding technique for computing similarity of text documents.

Next we explain how to instantiate chooseFakeRef and chooseFakeProbe. Let $\mathsf{chooseFakeRef}() := \mathfrak{b} := \begin{pmatrix} 0 & \ldots & 0 \end{pmatrix}^\top \in \mathbb{Z}_q^n$. Then, finding a fake probe template with distance d, is the same as finding a vector $\begin{pmatrix} \mathfrak{b}'_1 & \ldots & \mathfrak{b}'_n \end{pmatrix}^\top$ s.t. $\sum_{i \in [n]} \mathfrak{b}'^2_i = d$. When n is at least 4, then the existence of such a vector is guaranteed by Lagrange's four square theorem, furthermore there exist efficient algorithms for computing these four values [22]. We define chooseFakeProbe as setting the first four coordinates of $\mathfrak{b}'$ as the output of the algorithm of [22] and setting the other coordinates to 0. Note that chooseFakeRef and chooseFakeProbe are only part of the simulator and never need to be actually implemented or executed.

Hamming Distance: Iris readings can be transformed into binary vectors where a small Hamming distance corresponds to similar irises, e.g. with techniques described in [5]. For computing the Hamming distance d_H we use the same encoding technique as Kim et al. [18]. We assume that our templates are binary vectors $\mathfrak{b} \in \mathfrak{B} := \{0,1\}^n$. Then the largest possible Hamming distance is n, so we define $\mathcal{D} := \{0, \ldots, n\}$. We require from the IPFE scheme that $\mathcal{X} = \mathbb{Z}_q^n$ and $\mathcal{Y} = \mathbb{Z}_q$, for $q > n$. Furthermore, we define $\mathsf{out}(\mathfrak{b}, \mathfrak{b}') := d_H(\mathfrak{b}, \mathfrak{b}')$. For

encoding a template $\mathfrak{b}$ we define the vector $\mathbf{b}$ via $b_i = 1$, if $\mathfrak{b}_i = 1$ and $b_i = -1$ if $\mathfrak{b}_i = 0$. For example $(1, 0, 1)^\top$ becomes $(1, -1, 1)^\top$. We can then define $\mathsf{encodeRef}(\mathfrak{b}) := \mathsf{encodeProbe}(\mathfrak{b}) := \mathbf{b}$. However, the inner-product $\langle \mathbf{b}, \mathbf{b}' \rangle$ is not equal to the Hamming distance $d_H(\mathfrak{b}, \mathfrak{b}')$. Let ip be the inner-product output by the $\mathsf{IPFE.Dec}$ algorithm. Then we define $\mathsf{FE2out}(\mathsf{ip}) := \max(\frac{n-\mathsf{ip}}{2}, 0)$, to convert the inner-product into the Hamming distance.

Defining the algorithms for the simulator is relatively simple. We define $\mathsf{chooseFakeRef}() := \mathfrak{b} := (1 \dots 1)^\top \in \mathfrak{B}$ and $\mathsf{chooseFakeProbe}(\mathfrak{b}, d)$ as the same vector with d of the entries flipped to 0.

Cosine Similarity: As noted in [2], the cosine similarity can also be used for biometric matching (e.g. for faces as in [20]) and can be easily computed as an inner-product. The cosine similarity of two vectors $\mathbf{v}, \mathbf{w}$ is defined as $d_C(\mathbf{v}, \mathbf{w}) := \frac{\langle \mathbf{v}, \mathbf{w} \rangle}{\|\mathbf{v}\| \|\mathbf{w}\|}$, whereby the largest possible cosine similarity is 1, so we define $\mathcal{D} := [0, 1] \subset \mathbb{R}$. Close vectors have a high cosine similarity. We assume that the templates are vectors in $\mathfrak{B} := \mathbb{Z}_m^n$, with a fixed public L2 norm l. We require $\mathcal{X} = \mathbb{Z}_q^n, \mathcal{Y} = \mathbb{Z}_q$, for $q > l^2$. We define $\mathsf{out}(\mathfrak{b}, \mathfrak{b}') := d_C(\mathfrak{b}, \mathfrak{b}')$ and $\mathsf{encodeRef}(\mathfrak{b}) := \mathsf{encodeProbe}(\mathfrak{b}) := \mathfrak{b}$. Then the cosine similarity can be computed as $d_C(\mathfrak{b}, \mathfrak{b}') = \mathsf{FE2out}(\langle \mathbf{b}, \mathbf{b}' \rangle) := \min(\frac{\langle \mathbf{b}, \mathbf{b}' \rangle}{l^2}, 1)$.

Defining $\mathsf{chooseFakeRef}$ and $\mathsf{chooseFakeProbe}$ such that the templates have the correct distance, satisfy the norm bound l and still have integer coordinates seems rather complex and we leave it as future work. Thus, our security proof does not cover the cosine similarity instantiation.

Checking if the Distance is Below a Threshold: The above constructions leak the distance of the biometric templates to the server, which can be avoided by using a construction of [5]. They use orthogonality functional encryption to check if the inner-product is equal to a certain value and further use it to check if the distance of two encrypted biometric templates is within a certain range i.e. below a threshold. Their technique works with both the Euclidean and the Hamming distance and can be used to instantiate our framework so that the server only learns the yes-no authentication result.

6 Implementation

In this section we will briefly describe our proof of concept implementation and the results of our performance tests. The source code is available at: `https://github.com/johanernst/ipfe-bio-auth`. This includes the Golang source code of the protocol and the performance tests, the Python code for creating the plots and the raw output data of the experiments. For the function hiding IPFE scheme we used the implementation of the scheme of [18] in the GOFE library described in [21]. We run the performance tests on a single thread of an Intel Core i5-10210U CPU on a laptop. As biometric templates we used random vectors with increasing number of elements. Each element is an integer value

between 0 and 255 and we used the instantiation for the squared Euclidean distance described in Section 5. The choice of parameters is motivated by [23] who constructed a neural network that embeds face images into Euclidean space and has a very high accuracy. They use float vectors with 128 entries and report that "...it can be quantized to 128-bytes without loss of accuracy."

For each of the different template lengths we performed 3 runs, where each run consisted of enrolling/authenticating 10 clients. Finally we took the average to get the running time of a single call to each of the algorithms. We also separately measured the time spent in the calls to the IPFE schemes with the same number of runs and clients. The results are depicted in Figure 8 and Table 1.

The experiments show that most algorithms are rather fast (below 0.25ms), only the enrolment algorithm of the client is a bit slower (about 1.4s), because the Setup algorithm of the underlying IPFE scheme [18] requires the inversion of a $n \times n$ matrix, where n is the length of the vector. However, the enrolment is only performed once and the 1.4 seconds (for templates with 128 entries as in [23]) are unlikely to significantly disturb the user experience. The time that the server needs to enrol a client is so low because the server does not have to run any IPFE operations. Furthermore, Table 1 shows that most of the running time is consumed by the IPFE scheme. This means that our protocol profits significantly from future improvements in the area of function hiding IPFE. Also our scheme only needs *single-key* function hiding IPFE which can probably be constructed much more efficiently. Therefore, instantiating our protocol with a *single-key* function hiding IPFE will likely also boost efficiency. For templates with 128 entries the size of the enrolment message is 16.47 KB and the size of the authentication message is 33.29 KB.

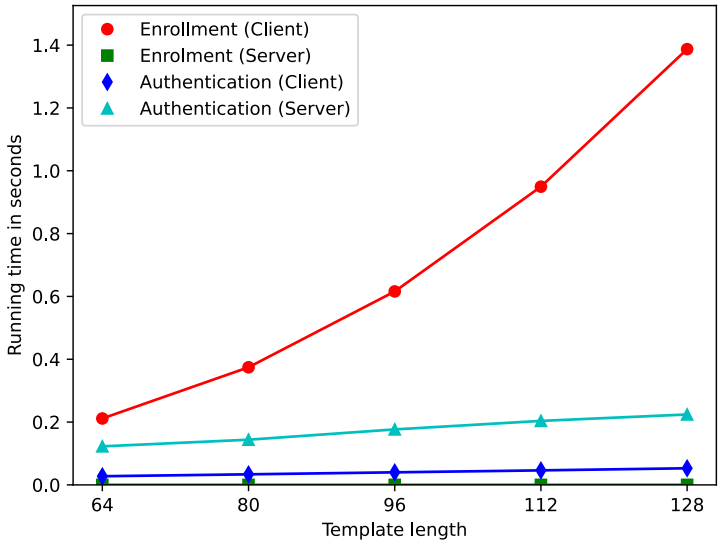

Fig. 8: The running time of the different parts of our protocol with different template lengths.

Table 1: This table shows the exact running times in seconds of the four algorithms of our protocol. In parenthesis there is the time spent in the calls to the IPFE scheme.

Template size	Client Enrolment	Server Enrolment	Client Authentication	Server Authentication
64	0.211 (0.210)	0.0001 (-)	0.028 (0.027)	0.123 (0.121)
128	1.387 (1.389)	0.0001 (-)	0.053 (0.054)	0.224 (0.224)

7 Conclusion and Future Work

Privacy-preserving biometric authentication is a complex problem and although significant work has been proposed in the area, the respective security models often do not capture all desired security goals or do not model all realistic adversarial behavior. In this paper we discussed and highlighted the importance of both *privacy preserving* and *secure* biometric authentication systems that maintain their security guarantees in complex real world settings. To this end we proposed an ideal functionality for universally composable biometric based two-factor authentication. To the best of our knowledge this is the first description of an ideal functionality for biometric based two-factor authentication in the UC framework. Furthermore, we proposed a general protocol for privacy preserving biometric authentication that provides UC security guarantees and can be instantiated using any suitable function hiding functional encryption scheme and a signature scheme. We provide a detailed security analysis and proof of the proposed general framework. Additionally, we showed how to concretely instantiate our framework with a function hiding IPFE scheme and, thereby, allow the computation of the Euclidean distance, the Hamming distance or the cosine similarity. Finally we explained our proof of concept implementation and presented the results of the performance tests.

Future Work: A worthwhile direction for future work may be to use our ideal functionality to analyze the FIDO2 protocol in the UC framework. The functionality would need to be adapted a little bit. For example it would probably need an interface for the server to indicate that they are willing to participate in the protocol and an interface for the adversary to allow the delivery of the server's message to the client.

Another interesting direction would be to extend our model to allow corrupted clients to get uncorrupted again. This can happen when the malware is removed from the client's device, or when the client gets back their, previously stolen, hardware token.

Acknowledgements This work was partially funded by the EU-funded Marie Curie ITN TReSPAsS-ETN project under the grant agreement 860813. We would like to thank the anonymous reviewers of ACNS for their detailed and helpful comments and suggestions and Astrid Ottenhues for helpful discussions.

References

[1] Shashank Agrawal et al. "BETA: Biometric-Enabled Threshold Authentication". In: *PKC 2021, Part II*. Ed. by Juan Garay. Vol. 12711. LNCS. Springer, Heidelberg, May 2021, pp. 290–318. DOI: 10.1007/978-3-030-75248-4_11.

[2] Shashank Agrawal et al. "Game-Set-MATCH: Using Mobile Devices for Seamless External-Facing Biometric Matching". In: *ACM CCS 2020*. Ed. by Jay Ligatti et al. ACM Press, Nov. 2020, pp. 1351–1370. DOI: 10.1145/3372297.3417287.

[3] Pia Bauspieß et al. "Post-Quantum Secure Two-Party Computation for Iris Biometric Template Protection". In: *2020 IEEE International Workshop on Information Forensics and Security (WIFS)*. IEEE. 2020, pp. 1–6.

[4] Julien Bringer, Hervé Chabanne, and Alain Patey. "Privacy-preserving biometric identification using secure multiparty computation: An overview and recent trends". In: *IEEE Signal Processing Magazine* 30.2 (2013), pp. 42–52.

[5] Chloe Cachet et al. *Proximity Searchable Encryption for Biometrics*. Cryptology ePrint Archive, Report 2020/1174. https://eprint.iacr.org/2020/1174. 2020.

[6] Ran Canetti. *Universally Composable Security: A New Paradigm for Cryptographic Protocols*. Cryptology ePrint Archive, Report 2000/067. https://eprint.iacr.org/2000/067. 2000.

[7] Jung Hee Cheon et al. "Lattice-Based Secure Biometric Authentication for Hamming Distance". In: *Australasian Conference on Information Security and Privacy*. Springer. 2021, pp. 653–672.

[8] John Daugman. "How iris recognition works". In: *The essential guide to image processing*. Elsevier, 2009, pp. 715–739.

[9] Jiankang Deng et al. "Arcface: Additive angular margin loss for deep face recognition". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 4690–4699.

[10] Pierre-Alain Dupont et al. "Fuzzy Password-Authenticated Key Exchange". In: *EUROCRYPT 2018, Part III*. Ed. by Jesper Buus Nielsen and Vincent Rijmen. Vol. 10822. LNCS. Springer, Heidelberg, 2018, pp. 393–424. DOI: 10.1007/978-3-319-78372-7_13.

[11] Andreas Erwig et al. "Fuzzy Asymmetric Password-Authenticated Key Exchange". In: *ASIACRYPT 2020, Part II*. Ed. by Shiho Moriai and Huaxiong Wang. Vol. 12492. LNCS. Springer, Heidelberg, Dec. 2020, pp. 761–784. DOI: 10.1007/978-3-030-64834-3_26.

[12] *FIDO specifications*. URL: https://fidoalliance.org/specifications/ (visited on 01/12/2023).

[13] Daniel Gardham, Mark Manulis, and Constantin Catalin Dragan. "Biometric-Authenticated Searchable Encryption". In: *ACNS 20, Part II*. Ed. by Mauro Conti et al. Vol. 12147. LNCS. Springer, Heidelberg, Oct. 2020, pp. 40–61. DOI: 10.1007/978-3-030-57878-7_3.

[14] Hasini Gunasinghe and Elisa Bertino. "PrivBioMTAuth: Privacy preserving biometrics-based and user centric protocol for user authentication from mobile phones". In: *IEEE Transactions on Information Forensics and Security* 13.4 (2017), pp. 1042–1057.

[15] Alberto Ibarrondo, Hervé Chabanne, and Melek Önen. "Practical Privacy-Preserving Face Identification based on Function-Hiding Functional Encryption". In: *International Conference on Cryptology and Network Security*. Springer. 2021, pp. 63–71.

[16] Stanislaw Jarecki, Hugo Krawczyk, and Jiayu Xu. "OPAQUE: An Asymmetric PAKE Protocol Secure Against Pre-computation Attacks". In: *EUROCRYPT 2018, Part III*. Ed. by Jesper Buus Nielsen and Vincent Rijmen. Vol. 10822. LNCS. Springer, Heidelberg, 2018, pp. 456–486. DOI: 10.1007/978-3-319-78372-7_15.

[17] Stanislaw Jarecki et al. "Two-Factor Authentication with End-to-End Password Security". In: *PKC 2018, Part II*. Ed. by Michel Abdalla and Ricardo Dahab. Vol. 10770. LNCS. Springer, Heidelberg, Mar. 2018, pp. 431–461. DOI: 10.1007/978-3-319-76581-5_15.

[18] Sam Kim et al. "Function-Hiding Inner Product Encryption Is Practical". In: *SCN 18*. Ed. by Dario Catalano and Roberto De Prisco. Vol. 11035. LNCS. Springer, Heidelberg, Sept. 2018, pp. 544–562. DOI: 10.1007/978-3-319-98113-0_29.

[19] Jascha Kolberg et al. "Template protection based on homomorphic encryption: Computationally efficient application to iris-biometric verification and identification". In: *2019 IEEE International Workshop on Information Forensics and Security (WIFS)*. IEEE. 2019, pp. 1–6.

[20] Weiyang Liu et al. "Sphereface: Deep hypersphere embedding for face recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 212–220.

[21] Tilen Marc et al. "Privacy-Enhanced Machine Learning with Functional Encryption". In: *ESORICS 2019, Part I*. Ed. by Kazue Sako, Steve Schneider, and Peter Y. A. Ryan. Vol. 11735. LNCS. Springer, Heidelberg, Sept. 2019, pp. 3–21. DOI: 10.1007/978-3-030-29959-0_1.

[22] Michael O Rabin and Jeffery O Shallit. "Randomized algorithms in number theory". In: *Communications on Pure and Applied Mathematics* 39.S1 (1986), S239–S256.

[23] Florian Schroff, Dmitry Kalenichenko, and James Philbin. "Facenet: A unified embedding for face recognition and clustering". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 815–823.

[24] Mei Wang et al. "Biometrics-Authenticated Key Exchange for Secure Messaging". In: *ACM CCS 2021*. Ed. by Giovanni Vigna and Elaine Shi. ACM Press, Nov. 2021, pp. 2618–2631. DOI: 10.1145/3460120.3484746.

[25] Kai Zhou and Jian Ren. "PassBio: Privacy-preserving user-centric biometric authentication". In: *IEEE Transactions on Information Forensics and Security* 13.12 (2018), pp. 3050–3063.

A Security Proof

Below we give the proof of Theorem 1.

Proof. We first provide a simulator in Figure 9 and Figure 10. Then we show that no PPT environment $\mathcal{Z}$ can distinguish between the real world, where it is interacting with the honest parties and the dummy adversary, from the ideal world, where it is interacting with the honest parties and the simulator. We do so by considering all actions that $\mathcal{Z}$ can take and argue for each of them that the results, which $\mathcal{Z}$ gets in the real world and the ideal world, are essentially the same. The actions that $\mathcal{Z}$ can take are:

- $(\textsc{Enrol}, \mathsf{sid}, \mathsf{ssid}, \mathfrak{b})$ to an honest client
- $(\textsc{Auth}, \mathsf{sid}, \mathsf{ssid}, \mathfrak{b}')$ to an honest client
- $(\textsc{ok}, (\mathsf{sid}, \mathsf{ssid}))$ to $\mathcal{F}'_{\text{SMT}}$
- $(\textsc{Send}, (\mathsf{sid}, \mathsf{ssid}), \mathcal{S}, m = (\text{enrol}, \mathsf{rid}, \mathsf{pk}, \mathsf{sk}_{\mathfrak{b}}))$ to $\mathcal{F}'_{\text{SMT}}$ in the name of a corrupted client
- $(\textsc{Send}, (\mathsf{sid}, \mathsf{ssid}), \mathcal{S}, m = (\text{auth}, \mathsf{rid}, c, \sigma))$ to $\mathcal{F}'_{\text{SMT}}$ in the name of a corrupted client
- $(\textsc{Corrupt}, \mathsf{sid})$ to a client $\mathcal{C}$
- $(\textsc{TryImpersonate}, \mathsf{sid}, \mathsf{ssid}, \mathfrak{b}')$ to a client $\mathcal{C}$

To simplify the presentation, we assume that $\mathcal{Z}$ does not delay or block messages, whenever the sender or receiver is corrupted. In that case the receiver directly gets the message without the need for $\mathcal{Z}$ to send $(\textsc{ok}, (\mathsf{sid}, \mathsf{ssid}))$ to $\mathcal{F}'_{\text{SMT}}$. This is reasonable, because $\mathcal{Z}$ cannot gain anything from blocking its own messages. Therefore, when $\mathcal{S}$ is corrupted, Sim can directly send $(\textsc{EnrolOK}, \mathsf{sid}, \mathsf{ssid})$ (resp. $(\textsc{AuthOK}, \mathsf{sid}, \mathsf{ssid})$) to $\mathcal{F}^{\mathsf{out}}_{\text{2FA}}$ after receiving $(\textsc{enrol}, \mathsf{sid}, \mathsf{ssid}, \cdot)$ (resp.$(\textsc{auth}, \mathsf{sid}, \mathsf{ssid}, \cdot)$) from $\mathcal{F}^{\mathsf{out}}_{\text{2FA}}$. In the real world we say that rid is *enroled*, if the server has a record $\langle \mathsf{rid}, \cdot, \cdot \rangle$. In the ideal world we say that rid is *enroled*, if the ideal functionality has a record $\langle \text{enroled}, \cdot, \mathsf{rid}, \cdot \rangle$ or $\langle \text{enroled-adversarial}, \mathsf{rid}, \cdot \rangle$. In both worlds this is equivalent to the server having output $(\textsc{enrol}, \mathsf{sid}, \mathsf{ssid}, \mathsf{rid})$, for some sid and ssid.

The simulator uses five different tables. Table T_1 is for pending messages, T_2 contains entries for the adversarially enroled clients. In case the server is corrupted, T_3 contains an entry for each of the enroled clients. Table T_4 contains an entry for each client that was adaptively corrupted by $\mathcal{Z}$ and T_5 contains all (fake) messages that Sim created as response to TryImpersonate instructions from $\mathcal{Z}$.

• $(\textsc{Enrol}, \mathsf{sid}, \mathsf{ssid}, \mathfrak{b})$ **to an honest client** $\mathcal{C}$: $\mathcal{Z}$ calls the enrol-interface of $\mathcal{C}$.

Case 1. The server is honest:

Real world: $\mathcal{C}$ only continues if this is the first Enrol message they got. They execute the setup algorithm of the FE scheme and the signature scheme and choose a random rid. $\mathcal{C}$ prepares the message m for the server and sends $(\textsc{Send}, (\mathsf{sid}, \mathsf{ssid}), \mathcal{S}, m)$ to $\mathcal{F}'_{\text{SMT}}$. $\mathcal{F}'_{\text{SMT}}$ then sends $(\textsc{sent}, (\mathsf{sid}, \mathsf{ssid}), \mathcal{C}, \mathcal{S}, \text{length}(m))$ to $\mathcal{A}$, who gives it to $\mathcal{Z}$.

Let l_e, l_a be the length of the enrolment- and authentication messages respectively.

Forwarding messages:

- on $(\text{ENROL}, \mathsf{sid}, \mathsf{ssid}, \mathcal{C})$ from $\mathcal{F}_{\text{2FA}}^{\text{out}}$:
  - `if` $\mathcal{S}$ is corrupted: send $(\text{ENROLOK}, \mathsf{sid}, \mathsf{ssid})$ to $\mathcal{F}_{\text{2FA}}^{\text{out}}$
  - `else`:
    - simulate $\mathcal{F}'_{\text{SMT}}$ by sending $(\text{SENT}, (\mathsf{sid}, \mathsf{ssid}), \mathcal{C}, \mathcal{S}, l_e)$ to $\mathcal{Z}$ as message from $\mathcal{F}'_{\text{SMT}}$
    - store $(\text{enrol}, \mathsf{ssid})$ in table T_1
- on $(\text{AUTH}, \mathsf{sid}, \mathsf{ssid}, \mathcal{C})$ from $\mathcal{F}_{\text{2FA}}^{\text{out}}$:
  - `if` $\mathcal{S}$ is corrupted: send $(\text{AUTHOK}, \mathsf{sid}, \mathsf{ssid})$ to $\mathcal{F}_{\text{2FA}}^{\text{out}}$
  - `else`:
    - simulate $\mathcal{F}'_{\text{SMT}}$ by sending $(\text{SENT}, (\mathsf{sid}, \mathsf{ssid}), \mathcal{C}, \mathcal{S}, l_a)$ to $\mathcal{Z}$ as message from $\mathcal{F}'_{\text{SMT}}$
    - store $(\text{auth}, \mathsf{ssid})$ in table T_1
- on $(\text{OK}, (\mathsf{sid}, \mathsf{ssid}))$ from $\mathcal{Z}$ to $\mathcal{F}'_{\text{SMT}}$:
  - `if` there is a record $(\text{enrol}, \mathsf{ssid})$ in table T_1: send $(\text{ENROLOK}, \mathsf{sid}, \mathsf{ssid})$ to $\mathcal{F}_{\text{2FA}}^{\text{out}}$
  - `else if` there is a record $(\text{auth}, \mathsf{ssid})$ in table T_1: send $(\text{AUTHOK}, \mathsf{sid}, \mathsf{ssid})$ to $\mathcal{F}_{\text{2FA}}^{\text{out}}$
  - `else:` ignore this message

Client messages to $\mathcal{F}'_{\text{SMT}}$:

- on $(\text{SEND}, (\mathsf{sid}, \mathsf{ssid}), \mathcal{S}, m = (\text{enrol}, \mathsf{rid}, \mathsf{pk}, \mathsf{sk}_{\mathfrak{b}}))$ from $\mathcal{Z}$ to $\mathcal{F}'_{\text{SMT}}$ (from corrupted client)
  - `if` server $\mathcal{S}$ is corrupted: send $(\text{SENT}, (\mathsf{sid}, \mathsf{ssid}), m)$ to $\mathcal{Z}$ as $\mathcal{F}'_{\text{SMT}}$'s output to $\mathcal{S}$
  - `else`:
    - choose $\mathfrak{b} \leftarrow \mathsf{chooseFakeRef}()$
    - send $(\text{ENROL}, \mathsf{sid}, \mathsf{ssid}, \mathsf{rid}, \mathfrak{b})$ to $\mathcal{F}_{\text{2FA}}^{\text{out}}$ //using the adversary's interface
    - `if` record $(\mathsf{rid}, \cdot, \cdot, \cdot)$ does *not* exist in table T_2: store $(\mathsf{rid}, \mathfrak{b}, \mathsf{pk}, \mathsf{sk}_{\mathfrak{b}})$ in table T_2
- on $(\text{SEND}, (\mathsf{sid}, \mathsf{ssid}), \mathcal{S}, m = (\mathsf{auth}, \mathsf{rid}, c, \sigma))$ from $\mathcal{Z}$ to $\mathcal{F}'_{\text{SMT}}$ (from corrupted client)
  - `if` server $\mathcal{S}$ is corrupted: send $(\text{SENT}, (\mathsf{sid}, \mathsf{ssid}), m)$ to $\mathcal{Z}$ as $\mathcal{F}'_{\text{SMT}}$'s output to $\mathcal{S}$
  - `else`:
    - `if` there is entry $(\mathsf{rid}, \mathfrak{b}, \mathsf{pk}, \mathsf{sk}_{\mathfrak{b}})$ in table T_2 and $\mathsf{Sig.Vfy}(\mathsf{pk}, \sigma, (\mathsf{sid}, \mathsf{ssid}, \mathsf{rid}, c)) = 1$:
      - $d := \mathsf{FE2out}(\mathsf{FE.Dec}(\mathsf{sk}_{\mathfrak{b}}, c)$
      - choose $\mathfrak{b}' \leftarrow \mathsf{chooseFakeProbe}(\mathfrak{b}, \mathsf{d})$ //make sure that $\mathsf{out}(\mathfrak{b}, \mathfrak{b}') = d$
      - send $(\text{AUTH}, \mathsf{sid}, \mathsf{ssid}, \mathsf{rid}, \mathfrak{b}')$ to $\mathcal{F}_{\text{2FA}}^{\text{out}}$ //using the adversary's interface
    - `else if` there is an entry $(\mathsf{rid}, \mathsf{sid}, \mathsf{ssid}, c, \mathsf{pk}, \mathcal{C}, \mathfrak{b}')$ in T_5 and $\mathsf{Sig.Vfy}(\mathsf{pk}, \sigma, (\mathsf{sid}, \mathsf{ssid}, \mathsf{rid}, c)) = 1$:
      - send $(\text{TRYIMPERSONATE}, \mathsf{sid}, \mathsf{ssid}, \mathcal{C}, \mathfrak{b}')$ to $\mathcal{F}_{\text{2FA}}^{\text{out}}$
    - `else`: send $(\text{AUTH}, \mathsf{sid}, \mathsf{ssid}, \perp, \perp)$ to $\mathcal{F}_{\text{2FA}}^{\text{out}}$ //using the adversary's interface

Simulating a corrupted server:

- on $(\text{ENROL}, \mathsf{sid}, \mathsf{ssid}, \mathsf{rid})$ from $\mathcal{F}_{\text{2FA}}^{\text{out}}$ to the corrupted server:
  - $\mathsf{msk} \leftarrow \mathsf{FE.Setup}(1^\lambda)$
  - choose $\mathfrak{b} \leftarrow \mathsf{chooseFakeRef}()$
  - $\mathbf{b} := \mathsf{encodeRef}(\mathfrak{b})$
  - $\mathsf{sk}_{\mathbf{b}} \leftarrow \mathsf{FE.KeyGen}(\mathsf{msk}, b)$
  - $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Sig.Gen}(1^\lambda)$
  - store $(\mathsf{rid}, \mathfrak{b}, \mathsf{pk}, \mathsf{sk}, \mathsf{msk}, \mathsf{sk}_{\mathbf{b}})$ in table T_3
  - send $(\text{SENT}, (\mathsf{sid}, \mathsf{ssid}), m = (\text{enrol}, \mathsf{rid}, \mathsf{pk}, \mathsf{sk}_{\mathbf{b}}))$ to $\mathcal{Z}$ as $\mathcal{F}'_{\text{SMT}}$'s output to $\mathcal{S}$
- on $(\text{AUTH}, \mathsf{sid}, \mathsf{ssid}, \mathsf{rid}, d)$ from $\mathcal{F}_{\text{2FA}}^{\text{out}}$ to the corrupted server:
  - retrieve record $(\mathsf{rid}, \mathfrak{b}, \mathsf{pk}, \mathsf{sk}, \mathsf{msk}, \mathsf{sk}_{\mathbf{b}})$ from table T_3
  - choose $\mathfrak{b}' \leftarrow \mathsf{chooseFakeProbe}(\mathfrak{b}, d)$ //make sure that $\mathsf{out}(\mathfrak{b}, \mathfrak{b}') = d$
  - $\mathbf{b}' := \mathsf{encodeProbe}(\mathfrak{b}')$
  - $c \leftarrow \mathsf{FE.Enc}(\mathsf{msk}, \mathbf{b}')$
  - $\sigma \leftarrow \mathsf{Sig.Sign}(\mathsf{sk}, (\mathsf{sid}, \mathsf{ssid}, \mathsf{rid}, c))$
  - send $(\text{SENT}, (\mathsf{sid}, \mathsf{ssid}), m = (\mathsf{auth}, \mathsf{rid}, c, \sigma))$ to $\mathcal{Z}$ as $\mathcal{F}'_{\text{SMT}}$'s output to $\mathcal{S}$

Fig. 9: The code of the simulator.

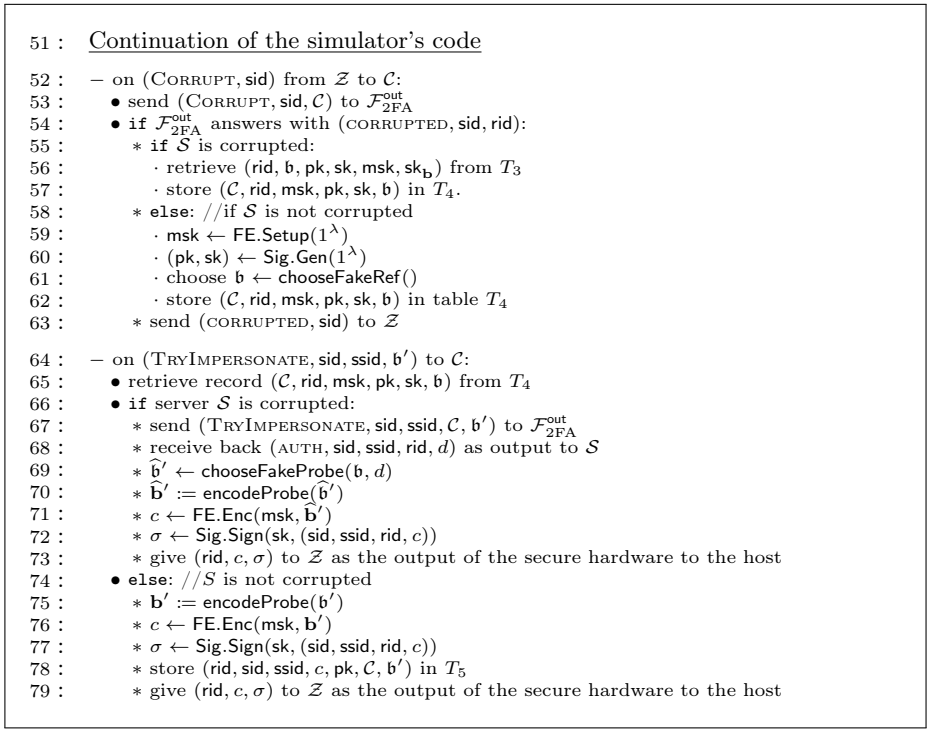

Fig. 10: The second part of the code of the simulator.

Ideal world: $\mathcal{C}$ sends $(\textsc{Enrol}, \mathsf{sid}, \mathsf{ssid}, \mathfrak{b})$ to $\mathcal{F}_{\text{2FA}}^{\mathsf{out}}$, which only continues if this is the first ENROL message from $\mathcal{C}$. $\mathcal{F}_{\text{2FA}}^{\mathsf{out}}$ then sends $(\textsc{enrol}, \mathsf{sid}, \mathsf{ssid}, \mathcal{C}, \mathcal{S})$ to Sim, who gives $(\textsc{sent}, (\mathsf{sid}, \mathsf{ssid}), \mathcal{C}, \mathcal{S}, l_e)$ to $\mathcal{Z}$.

In both worlds $\mathcal{Z}$ gets a message if and only if the client has not yet sent an enrolment message. By definition of l_e, we have that $\text{length}(m) = l_e$ and, therefore, the messages that $\mathcal{Z}$ gets in both worlds are the same.

Case 2. The server is corrupted:

Real world: $\mathcal{C}$ sends $(\textsc{Send}, (\mathsf{sid}, \mathsf{ssid}), \mathcal{S}, m := (\text{enrol}, \mathsf{rid}, \mathsf{pk}, \mathsf{sk_b}))$ to $\mathcal{F}'_{\text{SMT}}$ for random rid and fresh pk and $\mathsf{sk_b}$. $\mathcal{S}$ directly gives this message to $\mathcal{Z}$.

Ideal world: $\mathcal{F}_{\text{2FA}}^{\mathsf{out}}$ sends $(\textsc{enrol}, \mathsf{sid}, \mathsf{ssid}, \mathcal{C})$ to Sim, which replies with $(\textsc{EnrolOK}, \mathsf{sid}, \mathsf{ssid})$. Sim then gets $(\textsc{enrol}, \mathsf{sid}, \mathsf{ssid}, \mathsf{rid})$ as output to the corrupted server. Sim chooses $\mathfrak{b}$ and generates pk and $\mathsf{sk_b}$. They then give $(\textsc{sent}, (\mathsf{sid}, \mathsf{ssid}), m := (\text{enrol}, \mathsf{rid}, \mathsf{pk}, \mathsf{sk_b}))$ to $\mathcal{Z}$ in the name of the corrupted server.

In both worlds sid and ssid are the same and rid is a random value that $\mathcal{Z}$ has not previously seen. Also pk is in both worlds the result of $\mathsf{Sig.Gen}$. The only critical part is $\mathsf{sk_b}$. In the real world the underlying vector $\mathfrak{b}$ is the user's biometric, whereas in the ideal world the simulator chose $\mathfrak{b} \leftarrow \mathsf{chooseFakeRef}()$. However, both $\mathsf{sk_b}$ are indistinguishable due to the function hiding property of the FE scheme. In Lemma 1 we give a reduction that breaks the fh-IND-security of FE if $\mathcal{Z}$ can distinguish between the real and ideal world.

• $(\textsc{Auth}, \mathsf{sid}, \mathsf{ssid}, \mathfrak{b}')$ **to an honest client** $\mathcal{C}$: $\mathcal{Z}$ calls the auth-interface of $\mathcal{C}$.

Case 1. The server is honest:

Real world: $\mathcal{C}$ checks if they are enroled. If so, $\mathcal{C}$ prepares the authentication message m for the server and sends $(\textsc{Send}, (\mathsf{sid}, \mathsf{ssid}), \mathcal{S}, m)$ to $\mathcal{F}'_{\text{SMT}}$, which sends $(\textsc{sent}, (\mathsf{sid}, \mathsf{ssid}), \mathcal{C}, \mathcal{S}, \text{length}(m))$ to $\mathcal{A}$, who forwards it to $\mathcal{Z}$.

Ideal world: $\mathcal{C}$ sends $(\textsc{Auth}, \mathsf{sid}, \mathsf{ssid}, \mathfrak{b}')$ to $\mathcal{F}_{\text{2FA}}^{\mathsf{out}}$. If $\mathcal{C}$ has previously sent an enrol-message, $\mathcal{F}_{\text{2FA}}^{\mathsf{out}}$ sends $(\textsc{auth}, \mathsf{sid}, \mathsf{ssid}\mathcal{C}, \mathcal{S})$ to Sim, who gives $(\textsc{sent}, (\mathsf{sid}, \mathsf{ssid}), \mathcal{C}, \mathcal{S}, l_a)$ to $\mathcal{Z}$.

In both worlds $\mathcal{Z}$ gets a message if and only if the client has previously sent an enrolment message. By definition of l_a, we have that $\text{length}(m) = l_a$ and, therefore, the messages that $\mathcal{Z}$ gets are the same in both worlds.

Case 2. The server is corrupted:

Real world: If $\mathcal{C}$ has previously sent an enrol-message, they send $(\textsc{Send}, (\mathsf{sid}, \mathsf{ssid}), \mathcal{S}, m = (\text{auth}, \mathsf{rid}, c, \sigma))$ to $\mathcal{F}'_{\text{SMT}}$, where c is the encrypted, encoded $\mathfrak{b}$ and σ a signature of $(\mathsf{sid}, \mathsf{ssid}, \mathsf{rid}, c)$. $\mathcal{S}$ receives this message and directly gives it to $\mathcal{Z}$.

Ideal world: If $\mathcal{C}$ has previously sent an enrol-message, $\mathcal{F}_{\text{2FA}}^{\mathsf{out}}$ sends $(\textsc{auth}, \mathsf{sid}, \mathsf{ssid}, \mathcal{C})$ to Sim, which replies with $(\textsc{AuthOK}, \mathsf{sid}, \mathsf{ssid})$. Sim then gets $(\textsc{auth}, \mathsf{sid}, \mathsf{ssid}, \mathsf{rid}, d = \mathsf{out}(\mathfrak{b}, \mathfrak{b}'))$ as output to the corrupted server, where $\mathfrak{b}$ and $\mathfrak{b}'$ are the client's reference and fresh template. Sim chooses a fake probe template such that its FE output with the fake reference template is exactly d. Sim then encodes and encrypts the new fake template and creates a signature, using the self-chosen keys from the enrolment phase, as an honest client would do. They then give $(\textsc{sent}, (\mathsf{sid}, \mathsf{ssid}), m := (\text{auth}, \mathsf{rid}, c, \sigma))$ to $\mathcal{Z}$ in the name of the corrupted server.

In both worlds ssid is the same and rid is a random value that matches the rid from the enrolment phase. The critical component is the ciphertext c.

Here we rely on the fh-IND-security of the FE scheme, which ensures that no PPT adversary can distinguish between two ciphertexts, if the output of $\mathsf{FE.Dec}(\mathsf{sk_b}, \cdot)$ is the same in both cases. By choosing the fake templates as $\mathfrak{b}' \leftarrow \mathsf{chooseFakeProbe}(\mathfrak{b}, \mathsf{d})$, Sim ensures that the FE outputs are the same in both worlds. We show the indistinguishability of both worlds formally in Lemma 1 by giving a reduction, which breaks the fh-IND-security of the FE scheme, if $\mathcal{Z}$ is able to distinguish between the worlds. The signatures σ in both worlds are indistinguishable, as the secret keys are identically distributed and the signed messages are indistinguishable.

• $(\textbf{OK}, (\mathsf{sid}, \mathsf{ssid}))$ **to** $\mathcal{F}'_{\mathbf{SMT}}$: The environment lets through a client's message:

Case 1. (ssid belongs to an *enrol*-message of a client $\mathcal{C}$):

Real world: $\mathcal{C}$ chose rid uniformly from $\{0,1\}^\lambda$, therefore, $\mathcal{S}$ will not have a record $\langle \mathsf{rid}, \mathsf{pk}, \mathsf{sk_b} \rangle$ with overwhelming probability. Thus, $\mathcal{S}$ outputs (ENROL, sid, ssid, rid).

Ideal world: Sim sends (ENROLOK, $\mathsf{sid}, \mathsf{ssid}$) to $\mathcal{F}^{\mathsf{out}}_{\mathrm{2FA}}$. $\mathcal{F}^{\mathsf{out}}_{\mathrm{2FA}}$ will not have a record $\langle \mathrm{enroled}, \mathcal{C}, \cdot, \cdot \rangle$, because $\mathcal{C}$ has not yet enroled and enrols at most once. Hence, $\mathcal{F}^{\mathsf{out}}_{\mathrm{2FA}}$ will choose rid at random and give (ENROL, $\mathsf{sid}, \mathsf{ssid}, \mathsf{rid}$) as output to $\mathcal{Z}$.

In both worlds rid is a random bit string that $\mathcal{Z}$ has not seen before. Therefore, both worlds are indistinguishable for $\mathcal{Z}$.

Case 2. (ssid belongs to an *authentication*-message of a client $\mathcal{C}$): If $\mathcal{Z}$ previously let through the corresponding enrol-message of $\mathcal{C}$ then $\mathcal{S}$ has a record $\langle \mathsf{rid}, \mathsf{pk}, \mathsf{sk_b} \rangle$ and $\mathcal{F}^{\mathsf{out}}_{\mathrm{2FA}}$ has a record $\langle \mathrm{enroled}, \mathcal{C}, \mathsf{rid}, \mathfrak{b} \rangle$. Thus, in the real world $\mathcal{Z}$ will get $(\mathrm{AUTH}, \mathsf{sid}, \mathsf{ssid}, \mathsf{rid}, d = \mathsf{FE2out}(\mathsf{FE.Dec}(\mathsf{sk_b}, c)))$ from $\mathcal{S}$. In the ideal world $\mathcal{Z}$ will get $(\mathrm{AUTH}, \mathsf{sid}, \mathsf{ssid}, \mathsf{rid}, \mathsf{out}(\mathfrak{b}, \mathfrak{b}'))$, where $\mathfrak{b}, \mathfrak{b}'$ are the same in both worlds (chosen by $\mathcal{Z}$). In both worlds rid will match the rid from the enrol-message. By correctness of $(\mathsf{encodeRef}, \mathsf{encodeProbe}, \mathsf{out}, \mathsf{FE2out})$ we have $d = \mathsf{out}(\mathfrak{b}, \mathfrak{b}')$.

If $\mathcal{Z}$ did not let through $\mathcal{C}$'s enrol-message, $\mathcal{S}$ has no record $\langle \mathsf{rid}, \cdot, \cdot \rangle$ ($\mathcal{Z}$ does not even know rid) and $\mathcal{F}^{\mathsf{out}}_{\mathrm{2FA}}$ has no record $\langle \mathrm{enroled}, \mathcal{C}, \cdot, \cdot \rangle$. Thus, in both worlds, $\mathcal{Z}$ gets as output (AUTH-FAIL, $\mathsf{sid}, \mathsf{ssid}$).

• $(\textbf{SEND}, (\mathsf{sid}, \mathsf{ssid}), m = (\textbf{enrol}, \mathsf{rid}, \mathsf{pk}, \mathsf{sk_b}))$ **to** $\mathcal{F}'_{\mathbf{SMT}}$: A corrupted client's enrol-message:

Case 1. The server is honest:

Real world: If $\mathcal{S}$ previously output (ENROL, $\mathsf{sid}, \mathsf{ssid}', \mathsf{rid}$) (i.e. rid is already enroled), then $\mathcal{S}$ will output (ENROL, $\mathsf{sid}, \mathsf{ssid}, \perp$). Otherwise $\mathcal{S}$ will output (ENROL, $\mathsf{sid}, \mathsf{ssid}, \mathsf{rid}$).

Ideal world: Sim uses the adversary-interface of $\mathcal{F}^{\mathsf{out}}_{\mathrm{2FA}}$ by sending (ENROL, $\mathsf{sid}, \mathsf{ssid}$, $\mathsf{rid}, \mathfrak{b}$), for a fake template $\mathfrak{b}$. If rid is already enroled, $\mathcal{F}^{\mathsf{out}}_{\mathrm{2FA}}$ will output (ENROL, sid, $\mathsf{ssid}, \perp$) to $\mathcal{S}$ and otherwise (ENROL, $\mathsf{sid}, \mathsf{ssid}, \mathsf{rid}$). The outputs in both worlds are identical.

Case 2. The server is corrupted: In the real world $\mathcal{S}$ will receive (SENT, $(\mathsf{sid}, \mathsf{ssid}), m$) from $\mathcal{F}'_{\mathrm{SMT}}$ and output it to $\mathcal{Z}$. In the ideal world Sim will give (SENT, $(\mathsf{sid}, \mathsf{ssid}), m$) to $\mathcal{Z}$ in the name of $\mathcal{S}$. In both worlds $\mathcal{Z}$ gets identical output.

• $(\textbf{SEND}, (\mathsf{sid}, \mathsf{ssid}), m = (\textbf{auth}, \mathsf{rid}, c, \sigma))$ **to** $\mathcal{F}'_{\mathbf{SMT}}$: A corrupted client's auth-message:

Case 1. The server is honest: This is the case which shows that an attacker can first, not impersonate an honest client and second, still needs a valid biometric to impersonate an adaptively corrupted client.

First consider the case where rid is not enroled, or the signature σ is not valid. Then in both worlds $\mathcal{Z}$ will get $(\text{AUTH-FAIL}, \mathsf{sid}, \mathsf{ssid})$ as output from $\mathcal{S}$.

Next, consider the case that the signature is valid *and* rid belongs to a client that has been enroled by $\mathcal{A}$, i.e. $\mathcal{F}_{\text{2FA}}^{\mathsf{out}}$ has a record $\langle \text{enroled-adversarial}, \mathsf{rid}, \cdot \rangle$ and equivalently Sim has an entry $(\mathsf{rid}, \cdot, \cdot, \cdot)$ in T_2. In the real world, $\mathcal{S}$ will output $(\text{AUTH}, \mathsf{sid}, \mathsf{ssid}, \mathsf{rid}, d_{\text{real}})$ and in the ideal world $\mathcal{S}$ will output $(\text{AUTH}, \mathsf{sid}, \mathsf{ssid}, \mathsf{rid}, d_{\text{ideal}})$. We have $d_{\text{real}} = d_{\text{ideal}}$, because Sim computes its internal variable d exactly as the real server computes its output value and then uses it to generate fake templates that make $\mathcal{F}_{\text{2FA}}^{\mathsf{out}}$ output $\mathsf{out}(\mathfrak{b}, \mathfrak{b}')$ to $\mathcal{S}$. By correctness of $(\mathsf{chooseFakeRef}, \mathsf{chooseFakeProbe})$ Sim's fake template $\mathfrak{b}' \leftarrow \mathsf{chooseFakeProbe}(\mathfrak{b}, d)$ satisfies $\mathsf{out}(\mathfrak{b}, \mathfrak{b}') = d$.

Now consider the case where rid is enroled, the signature is valid *and* Sim has an entry $(\mathsf{rid}, \mathsf{sid}, \mathsf{ssid}, c, \cdot, \mathcal{C}, \mathfrak{b}')$ in T_5, where rid, sid, ssid and c are the same as from $\mathcal{Z}$'s message to $\mathcal{F}'_{\text{SMT}}$. This implies that $\mathcal{Z}$ has corrupted $\mathcal{C}$ and has sent a $(\text{TRYIMPERSONATE}, \mathsf{sid}, \mathsf{ssid}, \mathfrak{b}')$ instruction to $\mathcal{A}/\mathsf{Sim}$ and is now instructing $\mathcal{A}/\mathsf{Sim}$ to send the message —that $\mathcal{Z}$ got as response to the TRYIMPERSONATE instruction —to $\mathcal{S}$. Thus, in the real world $\mathcal{S}$ will output $(\text{AUTH}, \mathsf{sid}, \mathsf{ssid}, \mathsf{rid}, d_{\text{real}})$. In the ideal world Sim will send $(\text{TRYIMPERSONATE}, \mathsf{sid}, \mathsf{ssid}, \mathcal{C}, \mathfrak{b}')$ to $\mathcal{F}_{\text{2FA}}^{\mathsf{out}}$ which will then send $(\text{AUTH}, \mathsf{sid}, \mathsf{ssid}, \mathsf{rid}, \mathsf{out}(\mathfrak{b}, \mathfrak{b}'))$ to $\mathcal{S}$. Since in this case $\mathfrak{b}$ and $\mathfrak{b}'$ will be the same in both worlds, we have that $d_{\text{real}} = \mathsf{out}(\mathfrak{b}, \mathfrak{b}')$, by correctness of $(\mathsf{encodeRef}, \mathsf{encodeProbe}, \mathsf{out}, \mathsf{FE2out})$.

Let us now consider the last case, where neither of the above is true, Sim gets to the else-case (in line 34) *and* rid is enroled *and* the signature is valid. In the ideal world, Sim will send $(\text{AUTH}, \mathsf{sid}, \mathsf{ssid}, \perp, \perp)$ to $\mathcal{F}_{\text{2FA}}^{\mathsf{out}}$, which will then give $(\text{AUTH-FAIL}, \mathsf{sid}, \mathsf{ssid})$ as output to $\mathcal{S}$. In the real world, however, $\mathcal{S}$ will output $(\text{AUTH}, \mathsf{sid}, \mathsf{ssid}, \mathsf{rid}, d_{\text{real}})$. Thus, in this case $\mathcal{Z}$ can distinguish between the worlds. However, this case can only occur if $\mathcal{Z}$ forges a signature. In Lemma 2 we sketch a reduction that wins the EUF-CMA game in that case.

Case 2. The server is corrupted: Exactly as in the case of a corrupted client's enrol-message, in both worlds the server will output $(\text{SENT}, (\mathsf{sid}, \mathsf{ssid}), m)$.

• Instruction to $\mathcal{A}/\mathsf{Sim}$ to send $(\text{CORRUPT}, \mathsf{sid})$ to (the backdoor tape of) client $\mathcal{C}$:

Real world: $\mathcal{A}$ will send $(\text{CORRUPT}, \mathsf{sid})$ to $\mathcal{C}$ (on the backdoor tape). If and only if the client $\mathcal{C}$ exists *and* is enroled, $\mathcal{C}$'s shell will answer with $(\text{CORRUPTED}, \mathsf{sid})$. $\mathcal{A}$ will then forward this message to $\mathcal{Z}$.

Ideal world: Sim will send $(\text{CORRUPT}, \mathsf{sid}, \mathcal{C})$ to $\mathcal{F}_{\text{2FA}}^{\mathsf{out}}$. $\mathcal{F}_{\text{2FA}}^{\mathsf{out}}$ will answer with $(\text{CORRUPTED}, \mathsf{sid}, \mathsf{rid})$ if and only if the client $\mathcal{C}$ exists *and* is enroled. In that case Sim will send $(\text{CORRUPTED}, \mathsf{sid})$ to $\mathcal{Z}$.

Therefore, in both worlds $\mathcal{Z}$ will get the output $(\text{CORRUPTED}, \mathsf{sid})$ if and only if $\mathcal{C}$ exists *and* is enroled. This is independent of whether the server is corrupted.

• Instruction to $\mathcal{A}/\mathsf{Sim}$ to send $(\text{TRYIMPERSONATE}, \mathsf{sid}, \mathsf{ssid}, \mathfrak{b}')$ to (the backdoor tape of) client $\mathcal{C}$:

Case 1. The server is honest:
Real world: $\mathcal{A}$ will send (TryImpersonate, $\mathsf{sid}, \mathsf{ssid}, \mathfrak{b}'$) to $\mathcal{C}$ (on the backdoor tape). If $\mathcal{C}$ is corrupted, the shell will give (Auth, $\mathsf{sid}, \mathsf{ssid}, \mathfrak{b}'$) to the secure hardware, which will respond with $m = (\text{auth}, \mathsf{rid}, c, \sigma)$. The shell will give m to $\mathcal{A}$ who forwards it to $\mathcal{Z}$.
Ideal world: Sim will retrieve the fake keys from table T_4 which will exist only if $\mathcal{C}$ has been corrupted. Then Sim will encode, encrypt and sign $\mathfrak{b}'$ as the secure hardware would have done and give $m = (\text{auth}, \mathsf{rid}, c, \sigma)$ to $\mathcal{Z}$.

In both worlds $\mathcal{Z}$ will get an answer if and only if $\mathcal{C}$ has been corrupted before. In both worlds the rid is uniformly random, but stays the same over multiple TryImpersonate instructions. Furthermore, c and σ are generated with the same inputs and identically distributed keys which stay the same for multiple calls to TryImpersonate. Therefore, both worlds are perfectly indistinguishable.
Case 2. The server is corrupted:
Real world: $\mathcal{Z}$ will get the same as in the case of an uncorrupted server, namely $m = (\text{auth}, \mathsf{rid}, c, \sigma)$.
Ideal world: Sim will retrieve the fake keys from table T_4 which will exist only if $\mathcal{C}$ has been corrupted. Then Sim will send (TryImpersonate, $\mathsf{sid}, \mathsf{ssid}, \mathcal{C}, \mathfrak{b}'$) to $\mathcal{F}_{\text{2FA}}^{\mathsf{out}}$ and get back (auth, $\mathsf{sid}, \mathsf{ssid}, \mathsf{rid}, d$) as $\mathcal{F}_{\text{2FA}}^{\mathsf{out}}$'s answer to the corrupted server. Sim creates a fake probe template so that the distance to the earlier fake reference template is exactly d and encrypts and signs the message with the corresponding fake keys. Sim then gives $m = (\text{auth}, \mathsf{rid}, c, \sigma)$ to $\mathcal{Z}$.

In both worlds rid is uniformly random and stays the same over multiple calls to TryImpersonate. The encryption and signature keys are identically distributed and also stay the same for multiple calls to TryImpersonate. The only difference is that the ciphertext c in the real world is the encryption of $\mathfrak{b}'$, whereas in the ideal world it is the encryption of the fake probe template $\widehat{\mathfrak{b}'}$. In Lemma 1 we show that if $\mathcal{Z}$ can distinguish between the real and the ideal world, there is a reduction which breaks the fh-IND-security of the FE scheme.

Lemma 1. *If $\mathcal{Z}$ can distinguish between a key $\mathsf{sk}_{\mathfrak{b}}$ in the real world and the ideal world, or between a ciphertext c in the real world and the ideal world, then there is a reduction $\mathcal{B}$ that wins the fh-IND-security experiment of the FE scheme.*

Proof sketch. We use a hybrid argument over an upper bound on the number of honest clients l. Let H_i be the execution in which the first i honest clients use keys and ciphertexts as produced by the simulator. The other clients are still executed as in the real world. In a bit more detail, in H_i, for honest clients $\{1, \ldots, i\}$, whenever an enrolment-message is delivered to a corrupted server, $\mathcal{Z}$ gets the output of the "on (Enrol, $\mathsf{sid}, \mathsf{ssid}, \mathsf{rid}$) from $\mathcal{F}_{\text{2FA}}^{\mathsf{out}}$"-interface of the simulator. Whenever an authentication-message of one of the first i clients is delivered to a corrupted server, $\mathcal{Z}$ gets the output of the "on (Auth, $\mathsf{sid}, \mathsf{ssid}, \mathsf{rid}, d$) from $\mathcal{F}_{\text{2FA}}^{\mathsf{out}}$"-interface of the simulator. For clients $\{i+1, \ldots, l\}$, $\mathcal{Z}$ gets the output of the real clients Enrol (resp. Auth) interface. So in H_0 all secret keys $\mathsf{sk}_{\mathfrak{b}}$ and ciphertexts c are produced as in the real world, whereas in H_l all secret keys

and ciphertexts are produced as in the ideal world. An environment that is able to tell apart the real world from the ideal world by distinguishing between the real and simulated FE keys or ciphertexts, is also able to distinguish between H_0 and H_l. Therefore, there must exist $i \in \{1, \ldots, l\}$ such that $\mathcal{Z}$ can distinguish between H_{i-1} and H_i. We give a reduction $\mathcal{B}$ that wins the fh-IND-security experiment, given a distinguisher $\mathcal{D}$ for H_{i-1} and H_i:

When $\mathcal{Z}$ calls the "$(\textsc{Enrol}, \mathsf{sid}, \mathsf{ssid}, \mathfrak{b})$"-interface of the i-th honest client, $\mathcal{B}$ takes the public parameters from the fh-IND FE security experiment and asks a $\mathsf{QKeyGen}(\mathbf{b}, \widehat{\mathbf{b}})$ query, where $\mathbf{b}$ is the encoding of $\mathfrak{b}$ and $\widehat{\mathbf{b}} = \mathsf{encodeRef}(\mathsf{chooseFakeRef}())$ is the encoding of the fake reference template. $\mathcal{B}$ receives back the functional decryption key sk and gives this as part of the enrolment-message to the corrupted server and thereby to $\mathcal{Z}$. When $\mathcal{Z}$ calls the "$(\textsc{Auth}, \mathsf{sid}, \mathsf{ssid}, \mathfrak{b}')$"-interface of the i-th honest client, $\mathcal{B}$ asks a $\mathsf{QEnc}(\mathbf{b}', \widehat{\mathbf{b}'})$ query, where $\mathbf{b}'$ is the encoding of $\mathfrak{b}'$ and $\widehat{\mathbf{b}'}$ is the encoding of the fake probe template that the simulator would have chosen via $\mathsf{chooseFakeProbe}(\cdot, \cdot)$. $\mathcal{B}$ receives back the ciphertext c and gives this as part of the authentication-message to the corrupted server and thereby to $\mathcal{Z}$.

When the experiment's bit $b = 0$, then $\mathcal{B}$ gets the secret key and ciphertexts for the real biometric templates, whereby $\mathcal{B}$ perfectly simulates H_{i-1}. When the experiment's bit $b = 1$, then $\mathcal{B}$ gets the secret key and ciphertexts for the fake biometric templates chosen by the simulator, whereby $\mathcal{B}$ perfectly simulates H_i.

Lemma 2. *There is a reduction $\mathcal{B}$ that wins the EUF-CMA game if the environment manages to get to the else-case in line 34 of the simulator with a valid signature σ.*

Proof sketch. The general idea is that $\mathcal{B}$ runs the simulator's code, but whenever the simulator would create a signature keypair, or sign a message, $\mathcal{B}$ instead uses its challenger to get the keypair or signature.

A bit more in detail, $\mathcal{B}$ will guess a client $\mathcal{C}^*$. When Sim creates a keypair for that client in line 60 in Figure 10), $\mathcal{B}$ will get the public key from its EUF-CMA challenger. Whenever Sim would create a signature under the corresponding secret key (e.g. in line 77 in Figure 10), $\mathcal{B}$ asks a signing query to their challenger and uses the response as the signature that Sim would have created. When $\mathcal{B}$ gets a "$(\textsc{Send}, (\mathsf{sid}, \mathsf{ssid}), \mathcal{S}, m = (\mathsf{auth}, \mathsf{rid}, c, \sigma))$" instruction from $\mathcal{Z}$ with a valid signature σ (relative to the pk associated with rid), *and* gets to the else-case in line 34 in Figure 9), $\mathcal{B}$ outputs $((\mathsf{sid}, \mathsf{ssid}, \mathsf{rid}, c), \sigma)$ as forgery to its EUF-CMA challenger.

Now let us argue that this is indeed a valid forgery. First, observe that since $\mathcal{B}$ came to the else-case, it does not have an entry in table T_2, which means that the message did not belong to an adversarially enroled client and thereby pk was not chosen by $\mathcal{Z}$, but by $\mathcal{B}$'s EUF-CMA challenger. Second, since $\mathcal{B}$ came to the else-case, it also does not have a matching entry in table T_5, which means, it did not ask a signing query for $(\mathsf{sid}, \mathsf{ssid}, \mathsf{rid}, c)$ to its challenger in response to a TryImpersonate instruction. Therefore, $((\mathsf{sid}, \mathsf{ssid}, \mathsf{rid}, c), \sigma)$ constitutes a valid forgery and $\mathcal{B}$ wins the EUF-CMA game.