Two-Factor Authenticated Key Exchange with Enhanced Security from Post-Quantum Assumptions

Qijia Fan, Chenhao Bao, Xuanyu Shi, Shuai Han, and Shengli Liu

Shanghai Jiao Tong University, Shanghai 200240, China
{qweryy0566,bch123,shixuanyu,dalen17,slliu}@sjtu.edu.cn

Abstract. Two-factor authenticated key exchange (2fAKE) protocol requires a client to provide two authentication factors to authenticate itself and establish a shared session key with a server, thus providing double-insurance. Even if one factor is compromised, an adversary still cannot impersonate the client to pass the authentication. Popular choices of authentication factors include passwords, secret keys, biometrics, smart cards, tokens, etc.

In this work, we focus on 2fAKE with *biometrics* and *passwords* serving as the two authentication factors. Such 2fAKE might be the most convenient one for the client, who does not need to store any secrets or carry any additional device. On the flip side, the use of biometrics and passwords may increase the risk of their leakage. Considering that the biometrics of the client are unique and can hardly be changed, it is imminent to design 2fAKE which is able to protect the privacy of biometrics and passwords in the whole process of the protocol.

To this end, we formalize an *enhanced* security model for 2fAKE, which not only guarantees perfect forward security and mutually explicit authentication, but also ensures the privacy of biometrics and passwords. Specifically, we require *zero-knowledge* on the biometrics, namely that the databases of the server as well as the protocol transcripts do not reveal any information about the biometrics. As for passwords, we require that the compromise of the server database does not leak the passwords of the client.

We then propose a generic construction of 2fAKE, which is built on secure sketch, digital signature and asymmetric password-based AKE, in the random oracle model. Our 2fAKE enjoys both great convenience and enhanced security. It does not require the client to store any secrets, does not store the passwords in plain in the server database, and achieves zero-knowledge on the biometrics throughout the protocol. Furthermore, by instantiating the generic construction with post-quantum secure building blocks, we immediately get concrete 2fAKE protocol with enhanced security from *post-quantum assumptions*, which provides even stronger security guarantees. The experiments show that our 2fAKE instantiation achieves high efficiency, where the server stores just 3.1 KB in database for one client, the communication is only 3 rounds and costs 4.3 KB, the client runs in 25 ms and the server runs in only 0.2 ms.

Keywords: Two-factor authenticated key exchange, Biometrics, Passwords, Zero-Knowledge, Post-quantum security

1 Introduction

Authenticated Key Exchange (AKE) enables two parties to authenticate each other and establish a shared session key for secure communications. It is generally deployed in the client-server setting and enhances the security of various online services in our daily life, such as online shopping, online banking and online chatting. In these scenarios, a client authenticates himself/herself to the server by using some secret factor that only the client *knows*, like passwords or secret keys, or that the client *has*, like biometrics, smart cards, or tokens. Accordingly, these give rise to variants of AKE, like password-based AKE (PAKE) [4, 2, 30], public-key-infrastructure(PKI)-based AKE [3, 8, 33], biometric-based AKE [41, 25], etc. However, all these AKE variants fall into the category of *single-factor* AKE, and once the client's secret factor is leaked or compromised by an adversary, the security of AKE can no longer be guaranteed.

A good way to decrease the risk is using *Two-Factor* AKE (2fAKE) [22, 21, 20, 24, 45, 38], where the client now makes use of two secret factors to do authentication and key establishment with the server. Only if the client possesses both of the two factors, she/he can pass the authentication. Even if the adversary gets one factor of the client, it cannot impersonate the client to pass the authentication with the server. Thus 2fAKE provides stronger security guarantee and is preferable to single-factor AKE.

Different choices of the two factors lead to different types of 2fAKE, and among them, 2fAKE based on biometrics and passwords might be the most convenient one for the client. Biometrics (e.g., faces [34], fingerprints [44], palmprints [18], iris [13], voices [26]) are features of the client, and are naturally "carried" by the client without effort. In contrast, smart cards or tokens usually impose management burden on the client, who has to carry them and access them whenever needed. Besides, passwords are usually easier-to-remember than cryptographic secret keys [30]. With the combination of biometrics and passwords as two factors, the client does not need to carry any additional device or store them in device, and thus it provides a convenient and user-friendly manner to use 2fAKE.

However, the use of biometrics and passwords as two factors in 2fAKE might also increase the risk of their leakage. Biometrics are unique to the client and can hardly be changed, and once leaked, it would cause huge problems for the client. Moreover, in practice, the client tends to set the same password or related passwords for different services. Once one password is leaked, the security of other passwords would be compromised inevitably.

In summary, 2fAKE based on biometrics and passwords provides great convenience to the client, but on the other hand might increase the risk of leaking the factors. However, to the best of our knowledge, all existing biometric- and password-based 2fAKE schemes [38, 24, 20] do not take the protection of biometrics and passwords into account.

– The seminal work [38] by Pointcheval and Zimmer studies multi-factor AKE (a generalization of 2fAKE), where two of the multi-factors are biometrics

and passwords and the others can be secret keys, and formalizes the security for multi-factor AKE. However, their security model does not capture the protection of biometrics and passwords. Indeed, in their proposed scheme, the server stores the client's password in plain and encrypts the client's biometric using the client's key. If the server is compromised, the client's password is completely leaked, and moreover, if the adversary additionally gets the client's secret key, the client's biometric is also revealed by decryption. This is definitely unacceptable, since the leakage of secret key (one factor) should not lead to the leakage of biometric (another factor).

– The works [24] and [20] propose 2fAKE schemes, but they do not consider the protection of biometrics and passwords in their security model as well. Indeed, in the 2fAKE scheme [24] by Jiang et al., the server stores g^b in the database, where g is a generator of a cyclic group G and b denotes the client's biometric. Even though the discrete logarithm in G might be hard to solve in general, it is still possible that one can recover the client's biometric b from g^b, since the client's biometric b is not uniformly distributed[1]. Besides, the 2fAKE scheme [20] by Han et al. relies on trusted execution environment (TEE) for its security, which is vulnerable to side-channel attacks [15] and assumes completely trustworthy third-party (typically CPU manufacturer), thus limiting the application of their scheme.

Even worse, these existing schemes [38, 24, 20] all base their security on number-theoretic assumptions, such as the discrete logarithm or computational Diffie-Hellman (CDH) assumptions in cyclic groups, which are vulnerable to quantum adversaries. With the trend to migrate cryptographic algorithms to post-quantum ones (e.g., the NIST's post-quantum algorithm standardization [37]), it is desirable to build 2fAKE schemes from post-quantum assumptions. So it is natural to ask:

Can we construct biometric- and password-based 2fAKE which can additionally protect the privacy of biometrics and passwords, preferably based on post-quantum assumptions?

1.1 Our Contributions

In this paper, we answer the above question in the affirmative, by designing biometric- and password-based Two-Factor Authenticated Key Exchange (2fAKE) protocol with *enhanced* security and from *post-quantum* assumptions.

Enhanced Security Model for 2fAKE. Firstly, we establish an *enhanced* security model for 2fAKE, which provides three strong security guarantees: indistinguishability, authentication, and zero-knowledge. These three security notions are strong in the following sense:

[1] Recall that the well-accepted discrete logarithm problem is hard to solve only if the exponent b is uniformly distributed, e.g., see [29, Definition 8.62].

- *Indistinguishability* requires that the session keys established by the client and the server are pseudorandom (i.e., computationally indistinguishable from uniformly random keys), even if the adversary later corrupts both the client and server, thus providing *perfect forward security* [19, 31].
- *Authentication* guarantees that the client can authenticate herself/himself to the server only if she/he owns both of the two factors, and on the other hand, the server can authenticate itself to the client only if it possesses the enrollment data of the client, thus providing *mutually explicit authentication* [7, 40]. Besides, it further ensures that, even if the adversary compromises the database of the server and obtains the enrollment data of the client, the adversary still cannot impersonate the client to pass the authentication with the server. In particular, the compromise of the server database should not leak the passwords of the client. This reduces the impact of server database leakage on client authentication and thus provides a strong guarantee.
- *Zero-knowledge* ensures that the 2fAKE protocol computationally hides the biometrics of the client in the whole process of the protocol. More precisely, no matter for the client's enrollment data stored by the server, or for the transcripts of the 2fAKE protocol, all of them do not reveal any information about the biometrics of the client, thus providing *zero-knowledge on the client's biometrics*. Even if the adversary compromises the database of the server, and sees all the communications between the client and server, the adversary cannot learn any information about the biometrics of the client.

Overall, the enhanced security for 2fAKE provides perfect forward security, mutually explicit authentication, and additionally takes the protection of biometrics and passwords into account, so that even if the server database is compromised, the client's biometrics and passwords are still hidden to the adversary.

Generic Construction of 2fAKE. Next, we propose a generic framework for constructing 2fAKE protocol with enhanced security in the random oracle model. Our generic construction uses three building blocks, i.e., a secure sketch (SS) [12], a digital signature (SIG) and an augmented/asymmetric PAKE (aPAKE) [5, 16].

Roughly speaking, our generic 2fAKE works as follows (see also Fig. 3 in Sect. 4 for a graphical description):

– During the enrollment phase of 2fAKE (i.e., 2fAKE.Enroll in Fig. 3), the client generates a sampling w of her/his biometrics, and uses w to derive a signing key ssk of SIG and a hash value d via hash functions. Besides, the client uses her/his password pw to derive a password file rw according to the aPAKE protocol. Then the client sends the corresponding verification key vk, the password file rw and the hash value d to the server as the enrollment data.
– During the key exchange phase of 2fAKE (i.e., 2fAKE.Protocol in Fig. 3), the client and server first invoke the aPAKE protocol, using pw and rw as input respectively, and establish a shared aPAKE session key k^{aPAKE}. Then the server sends a hash value $t_{\mathsf{S}} = \mathsf{H}_{\mathsf{S}}(k^{\mathsf{aPAKE}} \| d)$ of k^{aPAKE} and d to the client to authenticate itself. Intuitively, the hash value $t_{\mathsf{S}} = \mathsf{H}_{\mathsf{S}}(k^{\mathsf{aPAKE}} \| d)$ functions as a proof of possession of rw and d, since without knowing rw, one cannot

impersonate the server and get k^{aPAKE}, guaranteed by the security of aPAKE. Accordingly, the client sends another hash value $t_{\mathsf{C}} = \mathsf{H}_{\mathsf{C}}(k^{\mathsf{aPAKE}})$ of k^{aPAKE} to the server to authenticate herself/himself, and similarly, $t_{\mathsf{C}} = \mathsf{H}_{\mathsf{C}}(k^{\mathsf{aPAKE}})$ behaves as a proof of possession of pw.

Besides, the client generates a fresh biometric sampling w', which is different from the sampling w generated during the enrollment phase but close to w. To recover w, the client employs SS to correct the errors incurred by the different biometric samplings w, w'. After getting w, the client can re-derive the signing key ssk, and use it to sign the transcripts of the protocol, yielding a signature σ. Then the client sends the signature σ to the server, who can verify the validity of σ with vk stored in the enrollment data. Intuitively, the signature σ behaves as a proof of possession of the client's biometrics, since without knowing a biometric sampling w', one can hardly re-derive the signing key ssk, and without getting ssk, one can hardly generate a valid signature σ, guaranteed by the security of SIG.

Finally, if the authentication checks pass, the client and server set a third hash value $k = \mathsf{H}_{\mathsf{key}}(k^{\mathsf{aPAKE}} \| d)$ of k^{aPAKE} and d as the session key of 2fAKE.

Intuitively, we rely on SS to make sure that the samplings w' of the biometric can be tuned to a unique sampling w, rely on SIG to authenticate the biometrics of the client, and rely on aPAKE to authenticate the password of the client as well as the enrollment data stored by the server. Overall, the client can pass the verification with the server (i.e., generating $t_{\mathsf{C}} = \mathsf{H}_{\mathsf{C}}(k^{\mathsf{aPAKE}})$ and valid σ) only if she/he owns both the biometrics and password pw, and the server can pass the verification with the client (i.e., generating $t_{\mathsf{S}} = \mathsf{H}_{\mathsf{S}}(k^{\mathsf{aPAKE}} \| d)$) only if it possesses rw and d contained in the enrollment data.

Moreover, we note that our proposed 2fAKE protocol achieves zero-knowledge on biometrics, since the server stores (vk, rw, d), which is just a verification key, a password file and a hash value, and the transcripts of 2fAKE include the transcripts of aPAKE together with two hash values $(t_{\mathsf{S}}, t_{\mathsf{C}})$ and a signature σ, all of which computationally hide the biometrics of the client in the random oracle model (cf. Sect. 5 for the formal proof).

Instantiations from Post-Quantum Assumptions. Finally, we obtain concrete 2fAKE protocol with enhanced security by instantiating our generic construction. Given that there are information-theoretical instantiations of SS [12, 27, 42, 43] and instantiations of SIG and aPAKE from post-quantum assumptions like SIS (short-integer-solution)/LWE (learning-with-errors) [17, 16], we immediately get 2fAKE protocol with enhanced security from post-quantum assumptions, which provides even stronger post-quantum security guarantees.

We implement our post-quantum secure 2fAKE protocol, and the experiments show that our concrete 2fAKE protocol achieves high efficiency, where the server stores just 3.1 KB in its database for one client, the communication is only 3 rounds and costs 4.3 KB, and the client runs in about 25 ms while the server runs in only 0.2 ms.

In Table 1, we compare our 2fAKE protocol with the existing biometric- and password-based 2fAKE schemes in terms of security and round efficiency.

Table 1. Comparison of our 2fAKE protocol with the existing 2fAKE schemes [38, 24, 20]. The columns **Indistinguishability?**, **Authentication?** and **Zero-Knowledge?** ask whether the scheme achieves the corresponding security, where zero-knowledge means that the scheme computationally hides the biometrics of clients, and "–" indicates that it is unknown whether the scheme achieves the security. The column **Assumption** shows the computational assumption on which the security is based. The column **Rounds** shows the rounds of interactions between the client and the server.

Schemes	Indistinguishability?	Authentication?	Zero-Knowledge?	Assumption	Rounds
[38]	✓	✓	×	CDH	4
[24]	✓	✓	×	CDH	4
[20]	✓	✓	–	CDH + TEE	6
Our 2fAKE	✓	✓	✓	LWE/SIS	3

2 Preliminaries

Notations. Let $\lambda \in \mathbb{N}$ denote the security parameter throughout the paper, and all algorithms, distributions, functions and adversaries take 1^λ as an implicit input. If x is defined by y or the value of y is assigned to x, we write $x := y$. For $n \in \mathbb{N}$, define $[n] := \{1, 2, ..., n\}$. For a set $\mathcal{X}$, denote by $x \leftarrow_\$ \mathcal{X}$ the procedure of sampling x from $\mathcal{X}$ uniformly at random. If $\mathcal{D}$ is distribution, $x \leftarrow_\$ \mathcal{D}$ means that x is sampled according to $\mathcal{D}$. For an algorithm $\mathcal{A}$, let $y \leftarrow \mathcal{A}(x; r)$ or simply $y \leftarrow \mathcal{A}(x)$ denote running $\mathcal{A}$ with input x and randomness r and assigning the output to y. "PPT" abbreviates probabilistic polynomial-time. Denote by $\mathsf{poly}(\lambda)$ some polynomial function and $\mathsf{negl}(\lambda)$ some negligible function in λ.

We present the definitions of secure sketch (SS), digital signature (SIG) and augmented/asymmetric password-based authenticated key exchange (aPAKE) in Appendix A, which will serve as the core building blocks in our generic construction of 2fAKE.

3 Two-Factor Authenticated Key Exchange (2fAKE)

In this section, we present the formal definition of Two-Factor Authenticated Key Exchange (2fAKE) protocol, through which clients utilize their *biometrics* (e.g., faces, fingerprints, palmprints, iris, voices) and *passwords* to realize authentication and establish session keys with a server.

More precisely, we define the syntax of 2fAKE in Subsect. 3.1 and formalize its enhanced security model in Subsect. 3.2.

3.1 Definition of 2fAKE

Roughly speaking, Two-Factor Authenticated Key Exchange (2fAKE) is invoked between two parties, namely a client C and a server S, and consists of two phases, i.e., the enrollment phase and the key exchange phase.

During the *enrollment* phase, C can utilize its own biometric W and password pw to produce a pair of strings $(\mathsf{edata}, \mathsf{pub})$. Here edata is an enrollment data that will be sent to S and stored in S's database, and pub is a helper string stored by C locally and publicly.

Then during the *key exchange* phase, C can use its two factors (W, pw) and the public helper string pub to execute an interactive key exchange protocol with S who possesses edata, and they can compute a shared session key for later use.

Below we present the formal definition and refer to Fig. 1 for an illustration of 2fAKE.

Definition 1 (Two-Factor Authenticated Key Exchange). *A Two-Factor Authenticated Key Exchange (2fAKE) protocol* $\mathsf{2fAKE} = (\mathsf{2fAKE.Setup}, \mathsf{2fAKE.Enroll}, \mathsf{2fAKE.Protocol})$ *consists of two PPT algorithms and a PPT interactive protocol.*

- $\mathsf{pp}_{\mathsf{2fAKE}} \leftarrow \mathsf{2fAKE.Setup}$*: The setup algorithm outputs a public parameter* $\mathsf{pp}_{\mathsf{2fAKE}}$*, which serves as an implicit input of other algorithms.*
- $(\mathsf{edata}, \mathsf{pub}) \leftarrow \mathsf{2fAKE.Enroll}(W, \mathsf{pw})$*: The enrollment algorithm takes as input a biometric* W *and a password* pw *of a client* C*, and outputs an enrollment data* edata *and a public helper string* pub*, where* edata *will be stored by a server* S *and* pub *will be stored by the client* C *itself.*
- $((\Psi_{\mathsf{C}}, k_{\mathsf{C}}), (\Psi_{\mathsf{S}}, k_{\mathsf{S}})) \leftarrow \mathsf{2fAKE.Protocol}(\mathsf{C}(\mathsf{res}_{\mathsf{C}}) \leftrightarrow \mathsf{S}(\mathsf{res}_{\mathsf{S}}))$*: The protocol is invoked by a client* C *and a server* S*, who have access to their own resources* $\mathsf{res}_{\mathsf{C}} := (W, \mathsf{pw}, \mathsf{pub})$ *and* $\mathsf{res}_{\mathsf{S}} := \mathsf{edata}$*, respectively. After the execution of the protocol, the client* C *outputs a flag* $\Psi_{\mathsf{C}} \in \{\emptyset, \textbf{\textit{accept}}, \textbf{\textit{reject}}\}$ *and a session key* $k_{\mathsf{C}} \in \mathcal{K} \cup \{\emptyset\}$ *(where* $\mathcal{K}$ *denotes the session key space), and similarly, the server* S *outputs* Ψ_{S} *and* k_{S}*.*

Correctness. If the client C and the server S execute the protocol $\mathsf{2fAKE.Protocol}$ honestly, they will both accept the session and output a same session key, i.e., $\Psi_{\mathsf{C}} = \Psi_{\mathsf{S}} = \textbf{accept}$ and $k_{\mathsf{C}} = k_{\mathsf{S}} \neq \emptyset$.

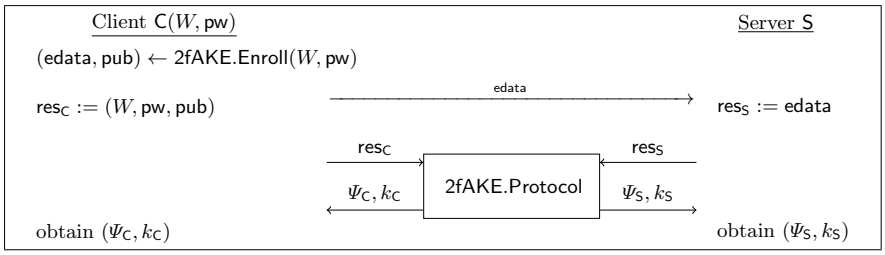

Fig. 1. A high-level illustration of 2fAKE.

3.2 Enhanced Security Model of 2fAKE

In this subsection, we formalize the enhanced security model for 2fAKE and define three important security notions, namely *indistinguishability, authentication, and zero-knowledge.* Roughly speaking,

- *Indistinguishability* requires that the session keys $k_{\mathsf{C}} = k_{\mathsf{S}}$ established by C and S are computationally indistinguishable from uniformly random keys.
- *Authentication* guarantees that S accepts a session only if C owns both of the two factors (W, pw), and on the other hand, C accepts a session only if S possesses edata.
- *Zero-knowledge* ensures that the strings edata and pub stored by S and C, as well as the transcripts of the protocol, do not reveal any information about the biometric W of C, and in fact, computationally hide it.

Our formalization follows a similar style to the existing security models for 2fAKE [38, 24, 20], and in particular, also uses the game-based definition in the Indistinguishability model (i.e., IND security), except that we additionally define the security notion of zero-knowledge to protect the privacy of biometric of clients.

More precisely, indistinguishability and authentication will be defined in a common security experiment (i.e., $\mathsf{Exp}_{\mathsf{2fAKE},\ell,\mathcal{A}}$ shown in Fig. 2), while zero-knowledge will be defined in another slightly different security experiment (i.e., $\mathsf{Exp}^{\mathsf{ZK}}_{\mathsf{2fAKE},\ell,\mathcal{A},\mathsf{Sim}}$ shown in Fig. 2). These two security experiments are mostly the same except for slight differences, so we will describe them together and then explain the differences. Firstly, these two security experiments are both played between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$ (except that the latter also involves a simulator Sim, which we will explain later). During the experiments, $\mathcal{C}$ will execute multiple 2fAKE protocol instances between the client C and the server S, while $\mathcal{A}$ can implement passive attacks as well as the following active attacks:

– *Corruption attacks*: $\mathcal{A}$ can implement three kinds of corruption attacks, including the corruption of the client C to obtain a sampling of its biometric W (i.e., corruption of biometric) or to obtain its password pw (i.e., corruption of password), or the compromise of the server S to obtain the enrollment data edata (i.e., corruption of enrollment data).
– *Session key reveal attacks*: $\mathcal{A}$ can obtain the session keys of some protocol instances.
– *Other active attacks*: $\mathcal{A}$ can interfere with all protocol instances by modifying, replacing, replaying, dropping, or injecting arbitrary messages.

Now we define oracles and their static variables in the security experiments.

Oracles and Security Experiments. Let $\mathsf{pp}_{\mathsf{2fAKE}} \leftarrow \mathsf{2fAKE.Setup}$ and $(\mathsf{edata}, \mathsf{pub}) \leftarrow \mathsf{2fAKE.Enroll}(W, \mathsf{pw})$. Suppose that C and S involve at most ℓ instances of the 2fAKE protocols. For party $U \in \{\mathsf{C}, \mathsf{S}\}$, U's involvement in these protocol instances can be formalized by a series of oracles $\pi^1_U, \cdots, \pi^\ell_U$, where

– Oracle π_U^i ($i \in [\ell]$) formalizes U's execution of the i-th protocol instance.

Each oracle π_{C}^i for C has access to the client's resources $\mathsf{res}_{\mathsf{C}} := (W, \mathsf{pw}, \mathsf{pub})$, and each oracle π_{S}^i for S has access to the server's resources $\mathsf{res}_{\mathsf{S}} := \mathsf{edata}$. Besides, each oracle π_U^i defines its own variables $\mathsf{var}_U^i := (k_U^i, \Psi_U^i)$ with initial values $(\emptyset, \emptyset)$.

– $k_U^i \in \mathcal{K} \cup \{\emptyset\}$: This variable records the session key computed by π_U^i.
– $\Psi_U^i \in \{\emptyset, \mathbf{accept}, \mathbf{reject}\}$: This variable indicates whether π_U^i has completed the protocol and accepted k_U^i. Note that $\Psi_U^i = \mathbf{accept}$ if and only if $k_U^i \neq \emptyset$.

We denote $\overline{U} := \{\mathsf{C}, \mathsf{S}\} \backslash \{U\}$ as the other party, i.e., $\overline{U} = \mathsf{C}$ if $U = \mathsf{S}$ and $\overline{U} = \mathsf{S}$ if $U = \mathsf{C}$.

Formally, $\mathcal{A}$ has access to the following oracles via queries.

– $\mathsf{Send}(U, i, m)$. It formalizes active/passive attacks implemented by $\mathcal{A}$. If $m = \top$, it means that $\mathcal{A}$ asks oracle π_U^i to send the first protocol message to $\overline{U}$. Otherwise, $\mathcal{A}$ impersonates $\overline{U}$ to send message m to π_U^i. Then π_U^i executes the 2fAKE protocol with m just like U does, computes a message m', and updates its own variables $\mathsf{var}_U^i = (k_U^i, \Psi_U^i)$. The output message m' is returned to $\mathcal{A}$.
  If $\mathsf{Send}(U, i, m)$ is the τ-th query asked by $\mathcal{A}$ and π_U^i changes Ψ_U^i to **accept** after that, then we say that π_U^i is τ-*accept*.
– CorruptBM. It formalizes the corruptions of C's biometric W. Upon this query, a sampling w of W is returned to $\mathcal{A}$.
– CorruptPW. It formalizes the corruption of C's password pw. Upon this query, pw is returned to $\mathcal{A}$.
  If $\mathcal{A}$ queries both CorruptBM (in the τ-th query) and CorruptPW (in the τ'-th query), we say that C is $\max\{\tau, \tau'\}$-*corrupted*. Otherwise, if $\mathcal{A}$ never queries CorruptBM or CorruptPW, we say that C is ∞-*corrupted*.
– CorruptED. It formalizes the corruption of the enrollment data edata. Upon this query, edata is returned to $\mathcal{A}$.
  If CorruptED is the τ-th query made by $\mathcal{A}$, we say that S is τ-*corrupted*. Otherwise, if $\mathcal{A}$ never queries CorruptED, we say that S is ∞-*corrupted*.
– $\mathsf{KReveal}(U, i)$. It formalizes session key reveal attacks implemented by $\mathcal{A}$, where $\mathcal{A}$ asks oracle π_U^i to reveal its session key k_U^i. If $\Psi_U^i \neq \mathbf{accept}$ (i.e., $k_U^i = \emptyset$), $\perp$ is returned to $\mathcal{A}$. Otherwise, the session key k_U^i is returned to $\mathcal{A}$.
– $\mathsf{Test}(U, i)$. This oracle is used to define indistinguishability (and also authentication) for 2fAKE, so it appears only in the first experiment (i.e., $\mathsf{Exp}_{\mathsf{2fAKE},\ell,\mathcal{A}}$ in Fig. 2). If $\Psi_U^i \neq \mathbf{accept}$ (i.e., $k_U^i = \emptyset$), the oracle returns $\perp$ to $\mathcal{A}$. Otherwise, the oracle sets $k_0 := k_U^i$, samples $k_1 \leftarrow_{\$} \mathcal{K}$, and return $k_{b_{\mathsf{Ind}}}$ to $\mathcal{A}$. Here $b_{\mathsf{Ind}} \leftarrow_{\$} \{0, 1\}$ is uniformly sampled challenge bit, chosen at the beginning of the experiment. The indistinguishability stipulates the hardness for $\mathcal{A}$ to guess the challenge bit b_{Ind} with probability non-negligibly better than $\frac{1}{2}$ (i.e., a random guess).

To characterize zero-knowledge on biometric for 2fAKE, we require that there exists a PPT simulator $\mathsf{Sim} = (\mathsf{Sim.Enroll}, \{\mathsf{Sim}.\pi_{\mathsf{C}}^i\}_{i \in [\ell]})$, where Sim.Enroll can

generate simulated $(\mathsf{edata}', \mathsf{pub}')$ and $\{\mathsf{Sim}.\pi_{\mathsf{C}}^i\}_{i\in[\ell]}$ can generate simulated transcripts of the protocol instances for C, without using C's biometric W. The simulated $(\mathsf{edata}', \mathsf{pub}')$ and transcripts are required to be indistinguishable from honestly generated ones. More precisely,

- $\mathsf{Sim.Enroll}(\mathsf{pw})$: It takes as input the client's password pw (but without the biometric W), and outputs a pair of simulated enrollment data edata' and public string pub'.
- $\mathsf{Sim}.\pi_{\mathsf{C}}^i$ $(i \in [\ell])$: These are simulated oracles that have access to $\mathsf{Sim.res_C} := (\mathsf{pw}, \mathsf{pub}')$ with pub' produced by $\mathsf{Sim.Enroll}$, but not to the biometric W, and generate simulated transcripts for C.

The security experiment for zero-knowledge is slightly different (see $\mathsf{Exp}^{\mathsf{ZK}}_{\mathsf{2fAKE},\ell,\mathcal{A},\mathsf{Sim}}$ in Fig. 2), where the challenger $\mathcal{C}$ will choose another challenge bit $b_{\mathsf{ZK}} \leftarrow_{\$} \{0,1\}$, and if $b_{\mathsf{ZK}} = 1$, $\mathcal{C}$ gives the simulated $(\mathsf{edata}', \mathsf{pub}')$ instead of the honestly generated $(\mathsf{edata}, \mathsf{pub})$ to $\mathcal{A}$, and uses the simulated oracles $\mathsf{Sim}.\pi_{\mathsf{C}}^i$ $(i \in [\ell])$ instead of the real oracles π_{C}^i $(i \in [\ell])$ to answer $\mathsf{Send}(U = \mathsf{C}, i, m)$ queries made by $\mathcal{A}$. The zero-knowledge requires the hardness for $\mathcal{A}$ to guess the challenge bit b_{ZK} with probability non-negligibly better than $\frac{1}{2}$.

To define the security notions of 2fAKE formally, trivial attacks must be identified and excluded. For instances, the tested session keys must not be revealed. We will describe trivial attacks later and first introduce the concepts of original key and partner according to [35].

Definition 2 (Original Key [35]). *For two oracles π_{C}^i and π_{S}^j, the original key, denoted as $\mathsf{K}(\pi_{\mathsf{C}}^i, \pi_{\mathsf{S}}^j)$ is the session key computed by the client and the server of the protocol under a passive adversary only.*

Definition 3 (Partner [35]). *Let $\mathsf{K}(\cdot,\cdot)$ denote the original key function. For two oracles π_{C}^i and π_{S}^j, we say that π_{C}^i is partnered to π_{S}^j, denoted as $\mathsf{Partner}(\pi_{\mathsf{C}}^i \leftarrow \pi_{\mathsf{S}}^j)$, if $k_{\mathsf{C}}^i = \mathsf{K}(\pi_{\mathsf{C}}^i, \pi_{\mathsf{S}}^j) \neq \emptyset$. Similarly, we say that π_{S}^j is partnered to π_{C}^i, denoted as $\mathsf{Partner}(\pi_{\mathsf{S}}^j \leftarrow \pi_{\mathsf{C}}^i)$, if $k_{\mathsf{S}}^j = \mathsf{K}(\pi_{\mathsf{C}}^i, \pi_{\mathsf{S}}^j) \neq \emptyset$. We write $\mathsf{Partner}(\pi_{\mathsf{C}}^i \leftrightarrow \pi_{\mathsf{S}}^j)$ if $\mathsf{Partner}(\pi_{\mathsf{C}}^i \leftarrow \pi_{\mathsf{S}}^j)$ and $\mathsf{Partner}(\pi_{\mathsf{S}}^j \leftarrow \pi_{\mathsf{C}}^i)$.*

Trivial Attacks. To clearly describe trivial attacks, we define the following flags.

- pwCorr: whether C's password pw is corrupted;
- bmCorr: whether C's biometric W is corrupted;
- edCorr: whether the enrollment data edata stored by S is corrupted;
- $\mathsf{Aflag}_{\mathsf{C}}^i$: whether the enrollment data edata stored by S is corrupted when π_{C}^i accepts;
- $\mathsf{Aflag}_{\mathsf{S}}^i$: whether both password pw and biometric W are corrupted when π_{S}^i accepts;
- Bflag^i: whether both password pw and biometric W are corrupted when π_{C}^i accepts;
- T_U^i: whether π_U^i is tested;

Table 2. Trivial Attacks **TA1-TA7** for the security experiments of 2fAKE. Here $\overline{U} := \{\mathsf{C}, \mathsf{S}\}\backslash\{U\}$ denotes the party other than U, i.e., $\overline{U} = \mathsf{C}$ if $U = \mathsf{S}$ and $\overline{U} = \mathsf{S}$ if $U = \mathsf{C}$.

Types	Trivial attacks	Explanation
TA1	$T^i_{\mathsf{C}} = \mathbf{true} \wedge \mathsf{Aflag}^i_{\mathsf{C}} = \mathbf{true}$	π^i_{C} is tested but S has the enrollment data edata revealed before π^i_{C} accepts session key k^i_{C}
TA2	$T^i_{\mathsf{S}} = \mathbf{true} \wedge \mathsf{Aflag}^i_{\mathsf{S}} = \mathbf{true}$	π^i_{S} is tested but C has both its biometric and password revealed before π^i_{S} accepts session key k^i_{S}
TA3	$T^i_{\mathsf{C}} = \mathbf{true} \wedge \mathsf{Bflag}^i = \mathbf{true}$	π^i_{C} is tested but C has both its biometric and password revealed before π^i_{C} accepts session key k^i_{C}
TA4	$T^i_U = \mathbf{true} \wedge \mathsf{kRev}^i_U = \mathbf{true}$	π^i_U is tested and its session key is revealed
TA5	$T^i_U = \mathbf{true}$ when $\mathsf{Test}(U, i)$ is queried	$\mathsf{Test}(U, i)$ is queried at least twice
TA6	$T^i_U = \mathbf{true} \wedge \mathsf{Partner}(\pi^i_U \leftrightarrow \pi^j_{\overline{U}})$ $\wedge \mathsf{kRev}^j_{\overline{U}} = \mathbf{true}$	π^i_U is tested, π^i_U and $\pi^j_{\overline{U}}$ are partnered to each other, and $\pi^j_{\overline{U}}$'s session key is revealed
TA7	$T^i_U = \mathbf{true} \wedge \mathsf{Partner}(\pi^i_U \leftrightarrow \pi^j_{\overline{U}})$ $\wedge T^j_{\overline{U}} = \mathbf{true}$	π^i_U is tested, π^i_U and $\pi^j_{\overline{U}}$ are partnered to each other, and $\pi^j_{\overline{U}}$ is tested

– kRev^i_U: whether the session key k^i_U is revealed.

Based on that we give a list of trivial attacks **TA1-TA7** in Table 2.

Actually, as the Test oracle appears only in the first security experiment (i.e., $\mathsf{Exp}_{\mathsf{2fAKE},\ell,\mathcal{A}}$), the variable T^i_U could only be set to **true** in the first security experiment, and consequently, the trivial attacks listed in Table 2 could only occur in the first security experiment. Putting differently, we in fact define no trivial attacks for the security experiment for zero-knowledge (i.e., $\mathsf{Exp}^{\mathsf{ZK}}_{\mathsf{2fAKE},\ell,\mathcal{A},\mathsf{Sim}}$).

Security Definition of 2fAKE. Based on the definitions of oracles and trivial attacks, we are ready to give the formal definition for the security notions of 2fAKE.

Definition 4 (Enhanced Security of 2fAKE). *Let ℓ be the maximum number of protocol executions, $\mathbf{W}$ the distribution of all possible biometrics, and $\mathcal{D}_{\mathsf{pw}}$ the password space.*

The security experiment $\mathsf{Exp}_{\mathsf{2fAKE},\ell,\mathcal{A}}$ for defining authentication and indistinguishability is described as follows (see also Fig. 2 with dotted boxes), which is played between challenger $\mathcal{C}$ and adversary $\mathcal{A}$.

1. *$\mathcal{C}$ invokes $\mathsf{2fAKE.Setup}$ to get $\mathsf{pp}_{\mathsf{2fAKE}}$, and chooses a biometric $W \leftarrow \mathbf{W}$ and a password $\mathsf{pw} \leftarrow_{\$} \mathcal{D}_{\mathsf{pw}}$ on behalf of the client. $\mathcal{C}$ also picks a challenge bit $b_{\mathsf{Ind}} \leftarrow_{\$} \{0, 1\}$.*
2. *Then $\mathcal{C}$ invokes $\mathsf{2fAKE.Enroll}(W, \mathsf{pw})$ to get the enrollment data edata and public helper string pub, and provides $\mathcal{A}$ with $\mathsf{pp}_{\mathsf{2fAKE}}$ and public data pub.*
3. *Next $\mathcal{A}$ issues queries to oracles $\mathsf{Send}, \mathsf{CorruptBM}, \mathsf{CorruptPW}, \mathsf{CorruptED}, \mathsf{KReveal}$ and Test adaptively. In particular,*
   – *If $b_{\mathsf{Ind}} = 0$, Test responds with real session keys; if $b_{\mathsf{Ind}} = 1$, Test responds with uniformly sampled session keys.*

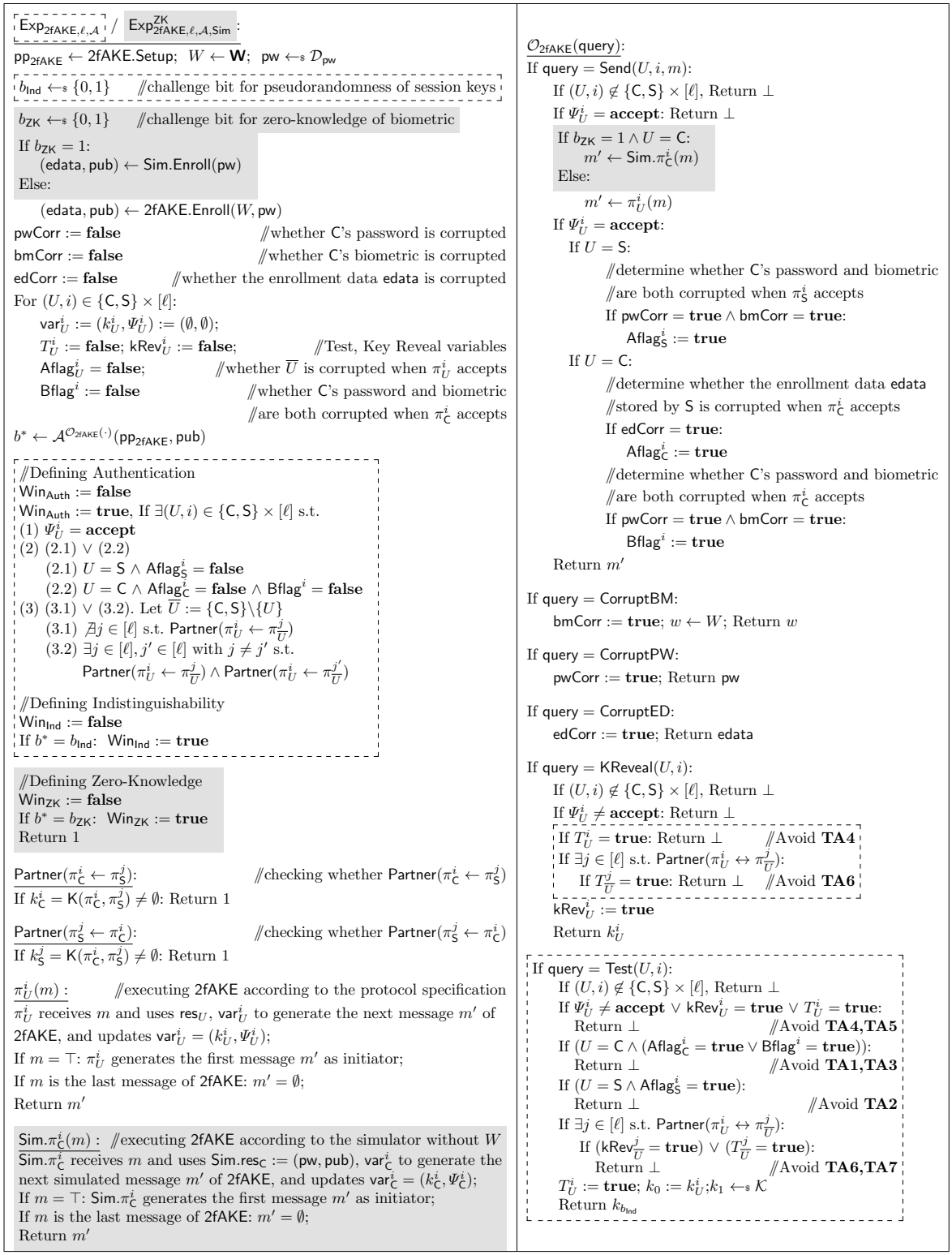

Fig. 2. The security experiment $\mathsf{Exp}_{\mathsf{2fAKE},\ell,\mathcal{A}}$ (with dotted boxes) for defining authentication and indistinguishability of 2fAKE, and the security experiment $\mathsf{Exp}^{\mathsf{ZK}}_{\mathsf{2fAKE},\ell,\mathcal{A},\mathsf{Sim}}$ (with gray boxes) for defining zero-knowledge of 2fAKE.

4. *Finally, $\mathcal{A}$ terminates with an output b^*.*

- ***Authentication.*** *Let* $\mathsf{Win}_{\mathsf{Auth}}$ *denote the event that $\mathcal{A}$ breaks authentication in the experiment* $\mathsf{Exp}_{\mathsf{2fAKE},\ell,\mathcal{A}}$. $\mathsf{Win}_{\mathsf{Auth}}$ *happens iff* $\exists(U,i) \in \{\mathsf{C},\mathsf{S}\} \times [\ell]$ *s.t.*
  (1) π_U^i *is τ-accepted;*
  (2) Either (2.1) or (2.2) happens.
  (2.1) $U = \mathsf{S}$, *and* $\overline{U} = \mathsf{C}$ *is $\hat{\tau}$-corrupted with $\hat{\tau} > \tau$.*
  (2.2) $U = \mathsf{C}$, $\overline{U} = \mathsf{S}$ *is $\hat{\tau}$-corrupted with $\hat{\tau} > \tau$, and* C *is $\hat{\tau}'$-corrupted with $\hat{\tau}' > \tau$.*
  (3) Either (3.1) or (3.2) happens.
  (3.1) There is no oracle $\pi_{\overline{U}}^j$ *that* π_U^i *is partnered to.*
  (3.2) There exist two oracles $\pi_{\overline{U}}^j$ *and* $\pi_{\overline{U}}^{j'}$ *with $j \neq j'$, to which* π_U^i *is partnered.*
  $\mathsf{2fAKE}$ *achieves authentication if for any* $\ell = \mathsf{poly}(\lambda)$, *any PPT adversary $\mathcal{A}$ launching at most Q queries to oracle* Send *(i.e., online dictionary attacks), we have*
  $$\mathsf{Adv}^{\mathsf{Auth}}_{\mathsf{2fAKE},\ell,\mathcal{A}} := \Pr[\mathsf{Win}_{\mathsf{Auth}}] \leq \mathsf{negl}(\lambda) + \tfrac{Q}{|\mathcal{D}_{\mathsf{pw}}|}.$$
- ***Indistinguishability.*** *Let* $\mathsf{Win}_{\mathsf{Ind}}$ *denote the event that* $b^* = b_{\mathsf{Ind}}$. $\mathsf{2fAKE}$ *achieves indistinguishability if for any* $\ell = \mathsf{poly}(\lambda)$ *and any PPT adversary $\mathcal{A}$ launching at most Q queries to oracle* Send *(i.e., online dictionary attacks), we have*
  $$\mathsf{Adv}^{\mathsf{Ind}}_{\mathsf{2fAKE},\ell,\mathcal{A}} := \left|\Pr[\mathsf{Win}_{\mathsf{Ind}}] - \tfrac{1}{2}\right| \leq \mathsf{negl}(\lambda) + \tfrac{Q}{|\mathcal{D}_{\mathsf{pw}}|}.$$

The security experiment $\mathsf{Exp}^{\mathsf{ZK}}_{\mathsf{2fAKE},\ell,\mathcal{A},\mathsf{Sim}}$ *for defining zero-knowledge is described as follows (see also Fig. 2 with gray boxes), played between challenger $\mathcal{C}$ and adversary $\mathcal{A}$.*

1. *$\mathcal{C}$ invokes* $\mathsf{2fAKE.Setup}$ *to get* $\mathsf{pp}_{\mathsf{2fAKE}}$, *and chooses a biometric* $W \leftarrow \mathbf{W}$ *and a password* $\mathsf{pw} \leftarrow_{\$} \mathcal{D}_{\mathsf{pw}}$ *on behalf of the client. $\mathcal{C}$ also picks a challenge bit* $b_{\mathsf{ZK}} \leftarrow_{\$} \{0,1\}$.
2. *If* $b_{\mathsf{ZK}} = 0$, *$\mathcal{C}$ invokes* $\mathsf{2fAKE.Enroll}(W, \mathsf{pw})$ *to get the enrollment data* edata *and public helper string* pub; *if* $b_{\mathsf{ZK}} = 1$, *$\mathcal{C}$ invokes* $\mathsf{Sim.Enroll}(\mathsf{pw})$ *to get the simulated* edata *and* pub *(here we omit the $'$ symbols), without using the biometric W of* C. *Then $\mathcal{C}$ provides $\mathcal{A}$ with* $\mathsf{pp}_{\mathsf{2fAKE}}$ *and public data* pub.
3. *Next $\mathcal{A}$ issues queries to oracles* $\mathsf{Send}, \mathsf{CorruptBM}, \mathsf{CorruptPW}, \mathsf{CorruptED}$ *and* $\mathsf{KReveal}$ *adaptively. In particular,*
   - *If* $b_{\mathsf{ZK}} = 0$ *or* $U = \mathsf{S}$, Send *responds by executing the real* $\mathsf{2fAKE}$ *protocol; if* $b_{\mathsf{ZK}} = 1$ *and* $U = \mathsf{C}$, Send *responds by executing the simulator* Sim *without using the biometric W of* C.
4. *Finally, $\mathcal{A}$ terminates with an output b^*.*

- ***Zero-Knowledge.*** *Let* $\mathsf{Win}_{\mathsf{ZK}}$ *denote the event that* $b^* = b_{\mathsf{ZK}}$. $\mathsf{2fAKE}$ *achieves zero-knowledge if for any* $\ell = \mathsf{poly}(\lambda)$ *and any PPT $\mathcal{A}$ that does not see the*

biometric W,[2] *there exists a PPT simulator* $\mathsf{Sim} = (\mathsf{Sim.Enroll}, \{\mathsf{Sim}.\pi_{\mathsf{C}}^i\}_{i\in[\ell]})$, *such that*

$$\mathsf{Adv}^{\mathsf{ZK}}_{\mathsf{2fAKE},\ell,\mathcal{A},\mathsf{Sim}} := \left|\Pr[\mathsf{Win}_{\mathsf{ZK}}] - \tfrac{1}{2}\right| \leq \mathsf{negl}(\lambda).$$

Remark. In the above definition, the term "$\frac{Q}{|\mathcal{D}_{\mathsf{pw}}|}$" captures the advantage of online dictionary attacks against the password. Online dictionary attacks are always possible since $\mathcal{A}$ can simply guess C's password pw uniformly at random, and the guess will be correct with probability $\frac{1}{|\mathcal{D}_{\mathsf{pw}}|}$ each time. Our definition essentially ensures that these online attacks are the only efficient attacks. Such term also appears in the security for existing 2fAKE schemes [24, 20].

4 Our 2fAKE Protocol

In this section, we present a generic construction of 2fAKE from a secure sketch SS, a digital signature scheme SIG, an augmented/asymmetric password-based authenticated key exchange protocol aPAKE and five hash functions $\mathsf{H}_{\mathsf{sig}}, \mathsf{H}_{\mathsf{d}}, \mathsf{H}_{\mathsf{S}}, \mathsf{H}_{\mathsf{C}}, \mathsf{H}_{\mathsf{key}}$.

More precisely, the underlying building blocks are as follows.

- Let $\mathsf{SS} = (\mathsf{SS.Gen}, \mathsf{SS.Rec})$ be an $(\mathcal{W}, m, \tilde{m}, t)$-secure sketch with simulatable sketches (cf. Def. 5 and Def. 6 in Appendix A).
- Let $\mathsf{SIG} = (\mathsf{SIG.Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ be a signature scheme with message space $\{0,1\}^*$, signing key space $\mathcal{SK}$, and canonical verification key generation algorithm $\mathsf{VK.Gen}$ (cf. Def. 7 in Appendix A).
- Let $\mathsf{aPAKE} = (\mathsf{aPAKE.Setup}, \mathsf{aPAKE.Enroll}, \mathsf{aPAKE.Protocol})$ be an aPAKE protocol (cf. Def. 8 in Appendix A).
- Let $\mathsf{H}_{\mathsf{sig}} : \mathcal{W} \times \{0,1\}^\lambda \to \mathcal{SK}$, $\mathsf{H}_{\mathsf{d}} : \mathcal{W} \times \{0,1\}^\lambda \to \{0,1\}^\lambda$, $\mathsf{H}_{\mathsf{S}} : \{0,1\}^* \to \{0,1\}^\lambda$, $\mathsf{H}_{\mathsf{C}} : \{0,1\}^* \to \{0,1\}^\lambda$ and $\mathsf{H}_{\mathsf{key}} : \{0,1\}^* \to \{0,1\}^\lambda$ be five hash functions.

Formally, our 2fAKE scheme is described as follows and shown in Fig. 3.

- $\underline{\mathsf{pp}_{\mathsf{2fAKE}} \leftarrow \mathsf{2fAKE.Setup}}$: The setup algorithm invokes $\mathsf{pp}_{\mathsf{aPAKE}} \leftarrow \mathsf{aPAKE.Setup}$ and returns the public parameters $\mathsf{pp}_{\mathsf{2fAKE}} := \mathsf{pp}_{\mathsf{aPAKE}}$.
- $\underline{(\mathsf{edata}, \mathsf{pub}) \leftarrow \mathsf{2fAKE.Enroll}(W, \mathsf{pw})}$: The enrollment algorithm takes as input a client's biometric W and password pw. It first samples the biometric with $w \leftarrow W$ and invokes $s \leftarrow \mathsf{SS.Gen}(w)$ to generate a sketch s. Then it samples a random string $x \leftarrow\!\!\$\ \{0,1\}^\lambda$, computes $ssk := \mathsf{H}_{\mathsf{sig}}(w, x)$ as its signing key, and invokes $vk := \mathsf{VK.Gen}(ssk)$ to get the corresponding verification key vk. Meanwhile, it computes $d := \mathsf{H}_{\mathsf{d}}(w, x)$, and invokes $\mathsf{rw} \leftarrow \mathsf{aPAKE.Enroll}(\mathsf{pw})$ to get rw. Finally, it returns $\mathsf{edata} := (vk, \mathsf{rw}, d)$ and $\mathsf{pub} := (s, x)$.

[2] This is reasonable, since if $\mathcal{A}$ already obtains a sampling w of W (e.g., by corrupting the client), it is meaningless to consider the zero-knowledge of biometric W.

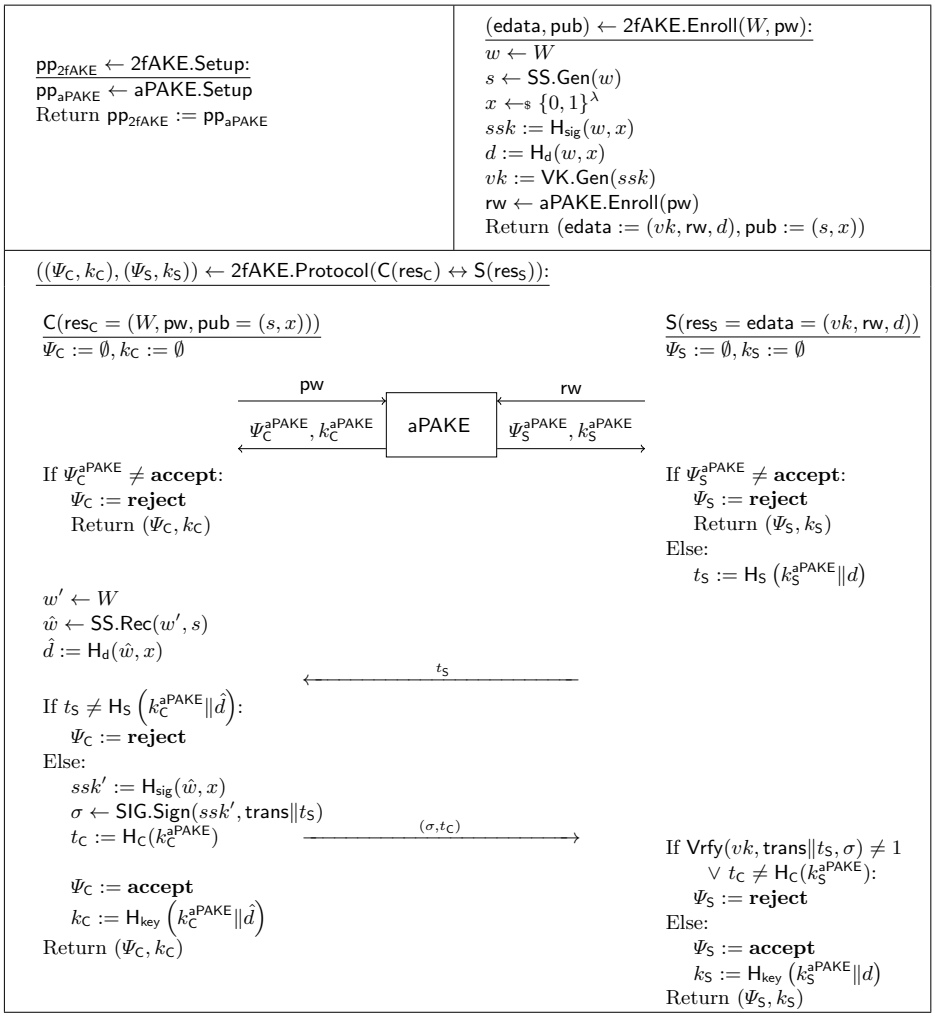

Fig. 3. Our generic construction of 2fAKE from secure sketch SS, signature scheme SIG, aPAKE protocol aPAKE and hash functions $\mathsf{H}_{\mathsf{sig}}, \mathsf{H}_{\mathsf{d}}, \mathsf{H}_{\mathsf{S}}, \mathsf{H}_{\mathsf{C}}, \mathsf{H}_{\mathsf{key}}$. Here trans denotes the transcripts of the underlying aPAKE protocol.

- $\underline{((\Psi_{\mathsf{C}}, k_{\mathsf{C}}), (\Psi_{\mathsf{S}}, k_{\mathsf{S}})) \leftarrow \mathsf{2fAKE.Protocol}(\mathsf{C}(\mathsf{res}_{\mathsf{C}}) \leftrightarrow \mathsf{S}(\mathsf{res}_{\mathsf{S}}))}$: The protocol is executed between a client C, who has access to resource $\mathsf{res}_{\mathsf{C}} := (W, \mathsf{pw}, \mathsf{pub} = (s, x))$ and a server S, who has access to resource $\mathsf{res}_{\mathsf{S}} := \mathsf{edata} = (vk, \mathsf{rw}, d)$. The execution consists of three stages:
  I. C and S first execute the aPAKE protocol $\mathsf{aPAKE.Protocol}(\mathsf{C}(\mathsf{pw}) \leftrightarrow \mathsf{S}(\mathsf{rw}))$ to get $((\Psi_{\mathsf{C}}^{\mathsf{aPAKE}}, k_{\mathsf{C}}^{\mathsf{aPAKE}}), (\Psi_{\mathsf{S}}^{\mathsf{aPAKE}}, k_{\mathsf{S}}^{\mathsf{aPAKE}}))$.
  For the client, if $\Psi_{\mathsf{C}}^{\mathsf{aPAKE}} \neq \mathbf{accept}$, C returns $\Psi_{\mathsf{C}} := \mathbf{reject}$ and terminates the execution. For the server, similarly, if $\Psi_{\mathsf{S}}^{\mathsf{aPAKE}} \neq \mathbf{accept}$, S returns $\Psi_{\mathsf{S}} := \mathbf{reject}$ and terminates the execution. Otherwise, if

$\Psi_{\mathsf{S}}^{\mathsf{aPAKE}} = \textbf{accept}$, S computes $t_{\mathsf{S}} := \mathsf{H}_{\mathsf{S}}(k_{\mathsf{S}}^{\mathsf{aPAKE}} \| d)$ and sends t_{S} to C, in order to authenticate itself to C.

II. After receiving t_{S}, C uses $\mathsf{pub} = (s, x)$ to regenerate the file d as follows, where s is the secure sketch and x is the random string. Namely, C samples its biometric with $w' \leftarrow W$, invokes $\hat{w} \leftarrow \mathsf{SS.Rec}(w', s)$, trying to reproduce w, and computes $\hat{d} := \mathsf{H}_{\mathsf{d}}(\hat{w}, x)$. Then C checks whether $t_{\mathsf{S}} = \mathsf{H}_{\mathsf{S}}(k_{\mathsf{C}}^{\mathsf{aPAKE}} \| \hat{d})$ holds. If $t_{\mathsf{S}} \neq \mathsf{H}_{\mathsf{S}}(k_{\mathsf{C}}^{\mathsf{aPAKE}} \| \hat{d})$, C returns $\Psi_{\mathsf{C}} := \textbf{reject}$ and terminates the execution.

Otherwise, C sets $\Psi_{\mathsf{C}} := \textbf{accept}$ and accepts $k_{\mathsf{C}} := \mathsf{H}_{\mathsf{key}}(k_{\mathsf{C}}^{\mathsf{aPAKE}} \| \hat{d})$ as its session key. Moreover, to authenticate itself to the server, C regenerates its signing key by computing $ssk' := \mathsf{H}_{\mathsf{sig}}(\hat{w}, x)$. With ssk', C signs the concatenation of the aPAKE transcripts trans in stage I and the received t_{S}, getting the corresponding signature $\sigma \leftarrow \mathsf{SIG.Sign}(ssk', \mathsf{trans} \| t_{\mathsf{S}})$. C also computes $t_{\mathsf{C}} := \mathsf{H}_{\mathsf{C}}(k_{\mathsf{C}}^{\mathsf{aPAKE}})$. Then C sends (σ, t_{C}) to the server S.

III. After receiving (σ, t_{C}), S checks whether σ is a valid signature w.r.t. vk, i.e., whether $\mathsf{Vrfy}(vk, \mathsf{trans} \| t_{\mathsf{S}}, \sigma) = 1$, and checks whether $t_{\mathsf{C}} = \mathsf{H}_{\mathsf{S}}(k_{\mathsf{S}}^{\mathsf{aPAKE}})$ holds. If the verification of σ fails or $t_{\mathsf{C}} \neq \mathsf{H}_{\mathsf{C}}(k_{\mathsf{S}}^{\mathsf{aPAKE}})$, S returns $\Psi_{\mathsf{S}} := \textbf{reject}$. Otherwise, it sets $\Psi_{\mathsf{S}} := \textbf{accept}$ and accepts $k_{\mathsf{S}} := \mathsf{H}_{\mathsf{key}}(k_{\mathsf{S}}^{\mathsf{aPAKE}} \| d)$ as its session key.

It is easy to check that the correctness of our 2fAKE protocol follows from the correctness of SS, SIG and aPAKE. Intuitively, we rely on SS to make sure that the samplings w' of a biometric W can be tuned to a unique sampling w. Then the hash function $\mathsf{H}_{\mathsf{sig}}$ converts w into a signing key ssk of SIG, and the hash function H_{d} converts w into a hash value d, which is used in the generation of t_{S} and session key $k_{\mathsf{C}} = k_{\mathsf{S}}$. We stress that the signing key ssk and the hash value d are never stored locally by the client. Instead, the client will regenerate them from its own biometric W whenever needed. Consequently, the client stores only a public string $\mathsf{pub} = (s, x)$ which helps in the regeneration of ssk and d.

5 Security Proof of Our 2fAKE Protocol

In this section, we establish the security of our 2fAKE protocol on the security of the underlying building blocks SS, SIG, aPAKE via the following theorem.

Theorem 1 (Security of Our 2fAKE). *Suppose that (i)* SS *is an* $(\mathcal{W}, m, \tilde{m}, t)$*-secure sketch with* $\tilde{m} = \omega(\log \lambda)$*, has simulatable sketches, and the samplings of biometric* W *have min-entropy at least* m*;*[3] *(ii)* SIG *is a secure digital signature scheme; (iii)* aPAKE *is a secure aPAKE protocol; (iv)* $\mathsf{H}_{\mathsf{sig}}, \mathsf{H}_{\mathsf{d}}, \mathsf{H}_{\mathsf{S}}, \mathsf{H}_{\mathsf{C}}, \mathsf{H}_{\mathsf{key}}$ *are modeled as random oracles. Then the* 2fAKE *proposed in Fig. 3 achieves indistinguishability, authentication, and zero-knowledge.*

Below we first give a proof sketch before presenting the formal proof.

[3] According to the discussion in Appendix A.1, it is easy to have simulatable sketches for secure sketch. By [9, 46], the biometrics like faces, fingerprints and iris have entropy of about 225 to 265 bits.

- *Authentication of our* 2fAKE*:* We will prove the authentication for the client C and the authentication for the server S separately.

  Authentication for C means that an adversary $\mathcal{A}$ can not impersonate C to make S accept a session, unless $\mathcal{A}$ corrupts both C's password pw (via CorruptPW query) and biometric W (via CorruptBM query). Roughly speaking, we show the authentication for C by considering two cases. In the case that $\mathcal{A}$ does not get pw, $\mathcal{A}$ can hardly impersonate C to execute aPAKE.Protocol with S according to the security of aPAKE, and thus the resulting $k_{\mathsf{S}}^{\mathsf{aPAKE}}$ is pseudorandom to $\mathcal{A}$. Consequently, $\mathcal{A}$ can hardly forge t_{C} satisfying $t_{\mathsf{C}} = \mathsf{H}_{\mathsf{C}}(k_{\mathsf{S}}^{\mathsf{aPAKE}})$. In the other case that $\mathcal{A}$ does not get a sampling of C's biometric W, the sampling w of W has enough average conditional entropy conditioned on the sketch s, so that the value of $ssk = \mathsf{H}_{\mathsf{sig}}(w, x)$ is uniformly random from $\mathcal{A}$'s view. Without knowing ssk, $\mathcal{A}$ can hardly produce a valid signature σ such that $\mathsf{Vrfy}(vk, \mathsf{trans} \| t_{\mathsf{S}}, \sigma) = 1$ holds, guaranteed by the security of SIG.

  Authentication for S means that an adversary $\mathcal{A}$ can not impersonate S to make C accept a session, unless $\mathcal{A}$ corrupts the enrollment data $\mathsf{edata} = (vk, \mathsf{rw}, d)$ stored by S (via CorruptED query), or corrupts both C's password pw (via CorruptPW query) and biometric W (via CorruptBM query). Similarly, we show the authentication for S by considering two cases. In the case that $\mathcal{A}$ does not get edata and pw, $\mathcal{A}$ can hardly impersonate S to execute aPAKE.Protocol with C according to the security of aPAKE, and thus the resulting $k_{\mathsf{C}}^{\mathsf{aPAKE}}$ is pseudorandom to $\mathcal{A}$. Consequently, $\mathcal{A}$ can hardly forge t_{S} satisfying $t_{\mathsf{S}} = \mathsf{H}_{\mathsf{S}}(k_{\mathsf{C}}^{\mathsf{aPAKE}} \| d)$. In the other case that $\mathcal{A}$ does not get edata and a sampling of C's biometric W, the sampling w of W has enough average entropy conditioned on the sketch s, and thus the value of $d = \mathsf{H}_{\mathsf{d}}(w, x)$ is uniformly random to $\mathcal{A}$. As a result, $\mathcal{A}$ can hardly forge t_{S} satisfying $t_{\mathsf{S}} = \mathsf{H}_{\mathsf{S}}(k_{\mathsf{C}}^{\mathsf{aPAKE}} \| d)$ either.

- *Indistinguishability of our* 2fAKE*:* If $\mathcal{A}$ corrupts C or S before a session key is accepted, the involved session key can not be tested, in order to avoid trivial attacks. On the other hand, if $\mathcal{A}$ does not corrupt C or S before a session key is accepted, by the authentication of our 2fAKE (as discussed above), $\mathcal{A}$ cannot impersonate C/S to make the other party accept. Consequently, for those tested session keys, $\mathcal{A}$ cannot interfere with the interactions between C and S, and in particular, cannot interfere with the executions of aPAKE.Protocol. There are two cases when C is not corrupted. If $\mathcal{A}$ does not have pw (and rw), then according to the security of aPAKE, the aPAKE session keys $k_{\mathsf{C}}^{\mathsf{aPAKE}} = k_{\mathsf{S}}^{\mathsf{aPAKE}}$ are pseudorandom to $\mathcal{A}$; if $\mathcal{A}$ has no samples of W, then according to the security of SS, the sampling w of W has enough average entropy conditioned on the sketch s, and thus the value of $d = \mathsf{H}_{\mathsf{d}}(w, x)$ is random to $\mathcal{A}$. In either case, the 2fAKE session keys $k_{\mathsf{C}} = \mathsf{H}_{\mathsf{key}}(k_{\mathsf{C}}^{\mathsf{aPAKE}} \| d) = \mathsf{H}_{\mathsf{key}}(k_{\mathsf{S}}^{\mathsf{aPAKE}} \| d) = k_{\mathsf{S}}$ are pseudorandom to $\mathcal{A}$. Consequently, the real session keys $k_0 = k_{\mathsf{C}} = k_{\mathsf{S}}$ are computationally indistinguishable from uniformly random keys $k_1 \leftarrow_{\$} \mathcal{K}$.

- *Zero-knowledge of our* 2fAKE*:* We will show that the enrollment data $\mathsf{edata} = (vk, \mathsf{rw}, d)$, the public helper string $\mathsf{pub} = (s, x)$, and the transcripts of our 2fAKE protocol computationally hide the biometric W of C.

  Let us first analyze $\mathsf{edata} = (vk, \mathsf{rw}, d)$ and $\mathsf{pub} = (s, x)$. Recall that they are generated by the 2fAKE.Enroll algorithm in the following way: $w \leftarrow W$, $s \leftarrow \mathsf{SS.Gen}(w)$, $x \leftarrow_{\$} \{0,1\}^{\lambda}$, $ssk := \mathsf{H}_{\mathsf{sig}}(w, x)$, $d := \mathsf{H}_{\mathsf{d}}(w, x)$, $vk := \mathsf{VK.Gen}(ssk)$ and $\mathsf{rw} \leftarrow \mathsf{aPAKE.Enroll}(\mathsf{pw})$. Note that if $\mathcal{A}$ never queries $\mathsf{H}_{\mathsf{sig}}$ on (w, x), then $ssk = \mathsf{H}_{\mathsf{sig}}(w, x)$ is just a uniformly random signing key and contains no information about w since $\mathsf{H}_{\mathsf{sig}}$ is modeled as random oracle, and so does vk. Indeed, $\mathcal{A}$ can hardly query $\mathsf{H}_{\mathsf{sig}}$ on (w, x), since conditioned on s, the sampling w has enough average conditional entropy according to the security of SS. Consequently, both ssk and vk contain no information about w. Similarly, d also contains no information about w. Moreover, rw and x do not involve w, and s can be efficiently simulated without knowing w. This shows the zero-knowledge of w in edata and pub.

  Next we analyze the transcripts of our 2fAKE protocol. Clearly, the transcripts of aPAKE involve only pw but not the biometric W. The only place where the biometric W is involved is that C uses W to regenerate ssk and d, and compute a signature $\sigma \leftarrow \mathsf{SIG.Sign}(ssk, \mathsf{trans} \| t_{\mathsf{S}})$. As we discussed above, ssk and d in fact contain no information about the biometric W, unless $\mathcal{A}$ can query $\mathsf{H}_{\mathsf{sig}}$ or H_{d} on (w, x), which can hardly occur. Consequently, the signature σ also contains no information about w. This shows the zero-knowledge of w in the transcripts of our 2fAKE.

  In the formal proof, we will build a simulator to simulate edata, pub and the transcripts of our 2fAKE without using the biometric W of C.

Next, we provide the formal proofs of authentication, indistinguishability and zero-knowledge in the following three subsections, respectively.

5.1 Proof of Authentication

Let $\mathcal{A}$ be an adversary that launches at most Q queries to oracle Send (i.e., online dictionary attacks). To prove that $\mathsf{Adv}^{\mathsf{Auth}}_{\mathsf{2fAKE},\ell,\mathcal{A}} = \Pr[\mathsf{Win}_{\mathsf{Auth}}] \leq \mathsf{negl}(\lambda) + \frac{Q}{|\mathcal{D}_{\mathsf{pw}}|}$, we divide the event $\mathsf{Win}_{\mathsf{Auth}}$ (cf. Def. 4) into two cases:

- **Case 1**: $\exists(U, i)$ s.t. (1) $\wedge$ (2.1) $\wedge$ (3) holds. Namely, $U = \mathsf{S}$, π^i_{S} is τ-accepted, $\overline{U} = \mathsf{C}$ is $\hat{\tau}$-corrupted with $\hat{\tau} > \tau$, but there is no oracle π^j_{C} or are two distinct oracles π^j_{C}, $\pi^{j'}_{\mathsf{C}}$ that π^i_{S} is partnered to. This captures the authentication for the client C.
- **Case 2**: $\exists(U, i)$ s.t. (1) $\wedge$ (2.2) $\wedge$ (3) holds. Namely, $U = \mathsf{C}$, π^i_{C} is τ-accepted, $\overline{U} = \mathsf{S}$ is $\hat{\tau}$-corrupted with $\hat{\tau} > \tau$, C is $\hat{\tau}'$-corrupted with $\hat{\tau}' > \tau$, but there is no oracle π^j_{S} or are two distinct oracles π^j_{S}, $\pi^{j'}_{\mathsf{S}}$ that π^i_{C} is partnered to. This captures the authentication for the server S.

We will show that Case 1 occurs with probability at most $\mathsf{negl}(\lambda) + \frac{Q_{\mathsf{S}}}{|\mathcal{D}_{\mathsf{pw}}|}$, and Case 2 occurs with probability at most $\mathsf{negl}(\lambda) + \frac{Q_{\mathsf{C}}}{|\mathcal{D}_{\mathsf{pw}}|}$, where Q_{S} denotes the number

of $\mathsf{Send}(U = \mathsf{S}, \cdot, \cdot)$ queries made by $\mathcal{A}$ and Q_{C} the number of $\mathsf{Send}(U = \mathsf{C}, \cdot, \cdot)$ queries. Clearly, $Q = Q_{\mathsf{S}} + Q_{\mathsf{C}}$.

Analysis of Case 1: Authentication for Client. Note that Case 1 means that the client C is not corrupted when π^i_{S} is accepted, i.e., either $\mathsf{CorruptPW}$ or $\mathsf{CorruptBM}$ has not been asked by the adversary $\mathcal{A}$. Consequently, $\mathcal{A}$ has not got the password pw or has not got a sampling of the biometric W of C at the time that π^i_{S} is accepted.

To show that $\Pr[\textbf{Case 1}] \leq \mathsf{negl}(\lambda) + \frac{Q_{\mathsf{S}}}{|\mathcal{D}_{\mathsf{pw}}|}$, we further divide Case 1 into two sub-cases and analyze them separately.

- **Case 1.1**: $\exists(U, i)$ s.t. $(1) \wedge (2.1) \wedge (3.1)$ holds. In this sub-case, there is no oracle π^j_{C} that π^i_{S} is partnered to. Consequently, it must be the adversary $\mathcal{A}$ who impersonated C to execute $\mathsf{aPAKE.Protocol}$ with S and forge (σ, t_{C}). However, the (σ, t_{C}) produced by $\mathcal{A}$ can hardly satisfy $\mathsf{Vrfy}(vk, \mathsf{trans} \| t_{\mathsf{S}}, \sigma) = 1$ and $t_{\mathsf{C}} = \mathsf{H}_{\mathsf{C}}(k^{\mathsf{aPAKE}}_{\mathsf{S}})$ simultaneously, as explained below, and thus π^i_{S} can hardly be accepted.

  If $\mathcal{A}$ has not queried $\mathsf{CorruptPW}$ to get the password pw, then after the execution of $\mathsf{aPAKE.Protocol}$ between $\mathcal{A}$ and S, the key $k^{\mathsf{aPAKE}}_{\mathsf{S}}$ established by S is pseudorandom to $\mathcal{A}$, guaranteed by the security of aPAKE. Consequently, $\mathcal{A}$ can hardly make a random oracle query on $k^{\mathsf{aPAKE}}_{\mathsf{S}}$, and the value of $\mathsf{H}_{\mathsf{C}}(k^{\mathsf{aPAKE}}_{\mathsf{S}})$ is uniformly random from $\mathcal{A}$'s view. So $\mathcal{A}$ can hardly output t_{C} such that $t_{\mathsf{C}} = \mathsf{H}_{\mathsf{C}}(k^{\mathsf{aPAKE}}_{\mathsf{S}})$ holds. The only exception is that $\mathcal{A}$ may guess the password pw itself without querying $\mathsf{CorruptPW}$. However, since $\mathcal{A}$ makes at most Q_{S} queries to $\mathsf{Send}(U = \mathsf{S}, \cdot, \cdot)$, and in each query, $\mathcal{A}$ can guess pw correctly with probability $\frac{1}{|\mathcal{D}_{\mathsf{pw}}|}$, this exception can happen with probability at most $\frac{Q_{\mathsf{S}}}{|\mathcal{D}_{\mathsf{pw}}|}$.

  If $\mathcal{A}$ has not queried $\mathsf{CorruptBM}$ to get a sampling of the biometric W, then the only information about W that $\mathcal{A}$ may obtain is the sketch s contained in pub. According to the security of SS, the sampling w of W has average conditional entropy of at least $\tilde{m} = \omega(\log \lambda)$ conditioned on the sketch s, so $\mathcal{A}$ can hardly make a random oracle query on (w, x), and the value of $ssk = \mathsf{H}_{\mathsf{sig}}(w, x)$ is uniformly random from $\mathcal{A}$'s view. Without knowing ssk, $\mathcal{A}$ can hardly produce a valid signature σ such that $\mathsf{Vrfy}(vk, \mathsf{trans} \| t_{\mathsf{S}}, \sigma) = 1$ holds, guaranteed by the security of SIG.

  Overall, $\mathcal{A}$ can hardly produce (σ, t_{C}) satisfying both $\mathsf{Vrfy}(vk, \mathsf{trans} \| t_{\mathsf{S}}, \sigma) = 1$ and $t_{\mathsf{C}} = \mathsf{H}_{\mathsf{C}}(k^{\mathsf{aPAKE}}_{\mathsf{S}})$, except with probability at most $\frac{Q_{\mathsf{S}}}{|\mathcal{D}_{\mathsf{pw}}|}$, and consequently, π^i_{S} can hardly be accepted. This shows that $\Pr[\textbf{Case 1.1}] \leq \mathsf{negl}(\lambda) + \frac{Q_{\mathsf{S}}}{|\mathcal{D}_{\mathsf{pw}}|}$.
- **Case 1.2**: $\exists(U, i)$ s.t. $(1) \wedge (2.1) \wedge (3.2)$ holds. In this sub-case, there are two distinct oracles π^j_{C} and $\pi^{j'}_{\mathsf{C}}$ that π^i_{S} is partnered to. This implies that $k^i_{\mathsf{S}} = \mathsf{K}(\pi^j_{\mathsf{C}}, \pi^i_{\mathsf{S}}) = \mathsf{K}(\pi^{j'}_{\mathsf{C}}, \pi^i_{\mathsf{S}}) \neq \emptyset$, i.e., $k^i_{\mathsf{S}} = \mathsf{H}_{\mathsf{key}}(k^{\mathsf{aPAKE},j}_{\mathsf{C}} \| d) = \mathsf{H}_{\mathsf{key}}(k^{\mathsf{aPAKE},j'}_{\mathsf{C}} \| d)$, where $k^{\mathsf{aPAKE},j}_{\mathsf{C}}$ (resp., $k^{\mathsf{aPAKE},j'}_{\mathsf{C}}$) denotes the aPAKE session key established by C in the j-th (resp., j'-th) protocol instance.

However, by the security of aPAKE, the aPAKE session keys $k_{\mathsf{C}}^{\mathsf{aPAKE},j}$ and $k_{\mathsf{C}}^{\mathsf{aPAKE},j'}$ in distinct protocol instances are pseudorandom and can hardly equal, so do their hash values $\mathsf{H}_{\mathsf{key}}(k_{\mathsf{C}}^{\mathsf{aPAKE},j}\|d)$ and $\mathsf{H}_{\mathsf{key}}(k_{\mathsf{C}}^{\mathsf{aPAKE},j'}\|d)$. Consequently, π_{S}^{i} can hardly be partnered to both π_{C}^{j} and $\pi_{\mathsf{C}}^{j'}$, and we get that $\Pr[\textbf{Case 1.2}] \leq \mathsf{negl}(\lambda)$.

Putting the above two sub-cases together, we have $\Pr[\textbf{Case 1}] \leq \Pr[\textbf{Case 1.1}] + \Pr[\textbf{Case 1.2}] \leq \mathsf{negl}(\lambda) + \frac{Q_{\mathsf{S}}}{|\mathcal{D}_{\mathsf{pw}}|}$. This shows the authentication for the client C.

Analysis of Case 2: Authentication for Server. Note that Case 2 means that both the server S and the client C are not corrupted when π_{C}^{i} is accepted, i.e., $\mathsf{CorruptED}$ is not queried by $\mathcal{A}$, and *either* $\mathsf{CorruptPW}$ *or* $\mathsf{CorruptBM}$ is not queried by $\mathcal{A}$ as well. Consequently, at the time that π_{C}^{i} is accepted, $\mathcal{A}$ has not got the enrollment data edata, and besides, $\mathcal{A}$ has not got *either* the password pw *or* a sampling of the biometric W.

To show that $\Pr[\textbf{Case 2}] \leq \mathsf{negl}(\lambda) + \frac{Q_{\mathsf{C}}}{|\mathcal{D}_{\mathsf{pw}}|}$, we further divide Case 2 into two sub-cases and analyze them separately.

- **Case 2.1**: $\exists(U, i)$ s.t. $(1) \wedge (2.2) \wedge (3.1)$ holds. In this sub-case, there is no oracle π_{S}^{j} that π_{C}^{i} is partnered to. Thus, it must be the adversary $\mathcal{A}$ who impersonated S to execute $\mathsf{aPAKE.Protocol}$ with C and forge t_{S}. However, the t_{S} produced by $\mathcal{A}$ can hardly satisfy $t_{\mathsf{S}} = \mathsf{H}_{\mathsf{S}}(k_{\mathsf{C}}^{\mathsf{aPAKE}}\|d)$, as explained below, so π_{C}^{i} can hardly be accepted.

  If $\mathcal{A}$ has queried neither $\mathsf{CorruptED}$ nor $\mathsf{CorruptPW}$ to get $\mathsf{edata} = (vk, \mathsf{rw}, d)$ and pw, then after the execution of $\mathsf{aPAKE.Protocol}$ between $\mathcal{A}$ and C, the key $k_{\mathsf{C}}^{\mathsf{aPAKE}}$ established by C is pseudorandom to $\mathcal{A}$, guaranteed by the security of aPAKE. Consequently, $\mathcal{A}$ can hardly make a random oracle query on $k_{\mathsf{C}}^{\mathsf{aPAKE}}\|d$, and the value of $\mathsf{H}_{\mathsf{S}}(k_{\mathsf{C}}^{\mathsf{aPAKE}}\|d)$ is uniformly random from $\mathcal{A}$'s view. So $\mathcal{A}$ can hardly output t_{S} s.t. $t_{\mathsf{S}} = \mathsf{H}_{\mathsf{S}}(k_{\mathsf{C}}^{\mathsf{aPAKE}}\|d)$ holds, and π_{C}^{i} can hardly be accepted. The only exception is that $\mathcal{A}$ may guess pw itself without querying $\mathsf{CorruptPW}$. However, since $\mathcal{A}$ makes at most Q_{C} queries to $\mathsf{Send}(U = \mathsf{C}, \cdot, \cdot)$, and in each query, $\mathcal{A}$ can guess pw correctly with probability $\frac{1}{|\mathcal{D}_{\mathsf{pw}}|}$, this exception can happen with probability at most $\frac{Q_{\mathsf{C}}}{|\mathcal{D}_{\mathsf{pw}}|}$.

  If $\mathcal{A}$ has queried neither $\mathsf{CorruptED}$ nor $\mathsf{CorruptBM}$ to get $\mathsf{edata} = (vk, \mathsf{rw}, d)$ and a sampling of the biometric W, then the only information about W that $\mathcal{A}$ may obtain is the sketch s contained in pub. According to the security of SS, the sampling w of W has average conditional entropy of at least $\tilde{m} = \omega(\log \lambda)$ conditioned on the sketch s, so $\mathcal{A}$ can hardly make a random oracle query on (w, x), and the value of $d = \mathsf{H}_{\mathsf{d}}(w, x)$ is uniformly random from $\mathcal{A}$'s view. As a result, $\mathcal{A}$ can hardly make a random oracle query on $k_{\mathsf{C}}^{\mathsf{aPAKE}}\|d$ (even if $k_{\mathsf{C}}^{\mathsf{aPAKE}}$ is known to $\mathcal{A}$), and the value of $\mathsf{H}_{\mathsf{S}}(k_{\mathsf{C}}^{\mathsf{aPAKE}}\|d)$ is also uniformly random from $\mathcal{A}$'s view. Similarly, π_{C}^{i} can hardly be accepted either.

Overall, $\mathcal{A}$ can hardly produce t_{S} satisfying $t_{\mathsf{S}} = \mathsf{H}_{\mathsf{S}}(k_{\mathsf{C}}^{\mathsf{aPAKE}} \| d)$, except with probability at most $\frac{Q_{\mathsf{C}}}{|\mathcal{D}_{\mathsf{pw}}|}$, and consequently, π_{C}^{i} can hardly be accepted. This shows that $\Pr[\textbf{Case 2.1}] \leq \mathsf{negl}(\lambda) + \frac{Q_{\mathsf{C}}}{|\mathcal{D}_{\mathsf{pw}}|}$.

– **Case 2.2**: $\exists(U, i)$ s.t. (1) $\wedge$ (2.2) $\wedge$ (3.2) holds. In this sub-case, there are two distinct oracles π_{S}^{j} and $\pi_{\mathsf{S}}^{j'}$ that π_{C}^{i} is partnered to. This implies that $k_{\mathsf{C}}^{i} = \mathsf{K}(\pi_{\mathsf{C}}^{i}, \pi_{\mathsf{S}}^{j}) = \mathsf{K}(\pi_{\mathsf{C}}^{i}, \pi_{\mathsf{S}}^{j'}) \neq \emptyset$, i.e., $k_{\mathsf{C}}^{i} = \mathsf{H}_{\mathsf{key}}(k_{\mathsf{S}}^{\mathsf{aPAKE},j} \| d) = \mathsf{H}_{\mathsf{key}}(k_{\mathsf{S}}^{\mathsf{aPAKE},j'} \| d)$, where $k_{\mathsf{S}}^{\mathsf{aPAKE},j}$ (resp., $k_{\mathsf{S}}^{\mathsf{aPAKE},j'}$) denotes the aPAKE session key established by S in the j-th (resp., j'-th) protocol instance.

  However, by the security of aPAKE, the aPAKE session keys $k_{\mathsf{S}}^{\mathsf{aPAKE},j}$ and $k_{\mathsf{S}}^{\mathsf{aPAKE},j'}$ in distinct protocol instances are pseudorandom and can hardly equal, so do their hash values $\mathsf{H}_{\mathsf{key}}(k_{\mathsf{S}}^{\mathsf{aPAKE},j} \| d)$ and $\mathsf{H}_{\mathsf{key}}(k_{\mathsf{S}}^{\mathsf{aPAKE},j'} \| d)$. Consequently, π_{C}^{i} can hardly be partnered to both π_{S}^{j} and $\pi_{\mathsf{S}}^{j'}$, and we get that $\Pr[\textbf{Case 2.2}] \leq \mathsf{negl}(\lambda)$.

Putting the above two sub-cases together, we have $\Pr[\textbf{Case 2}] \leq \Pr[\textbf{Case 2.1}] + \Pr[\textbf{Case 2.2}] \leq \mathsf{negl}(\lambda) + \frac{Q_{\mathsf{C}}}{|\mathcal{D}_{\mathsf{pw}}|}$. This shows the authentication for the server S.

Finally, taking all things together, we get that $\mathsf{Adv}^{\mathsf{Auth}}_{\mathsf{2fAKE},\ell,\mathcal{A}} = \Pr[\mathsf{Win}_{\mathsf{Auth}}] \leq \Pr[\textbf{Case 1}] + \Pr[\textbf{Case 2}] \leq \mathsf{negl}(\lambda) + \frac{Q_{\mathsf{S}}+Q_{\mathsf{C}}}{|\mathcal{D}_{\mathsf{pw}}|} = \mathsf{negl}(\lambda) + \frac{Q}{|\mathcal{D}_{\mathsf{pw}}|}$, so the authentication of 2fAKE follows. □

5.2 Proof of Indistinguishability

According to the proof of authentication in Subsect. 5.1, we know that $\mathsf{Win}_{\mathsf{Auth}}$ occurs with probability at most $\mathsf{negl}(\lambda) + \frac{Q}{|\mathcal{D}_{\mathsf{pw}}|}$ in the security experiment $\mathsf{Exp}_{\mathsf{2fAKE},\ell,\mathcal{A}}$.

To prove that $\mathsf{Adv}^{\mathsf{Ind}}_{\mathsf{2fAKE},\ell,\mathcal{A}} = \left|\Pr[\mathsf{Win}_{\mathsf{Ind}}] - \frac{1}{2}\right| \leq \mathsf{negl}(\lambda) + \frac{Q}{|\mathcal{D}_{\mathsf{pw}}|}$, it suffices to show that when $\mathsf{Win}_{\mathsf{Auth}}$ does not happen, for all $\mathsf{Test}(U, i)$ queries made by $\mathcal{A}$, the real session keys $k_0 = k_U^i$ are computationally indistinguishable from uniformly random keys $k_1 \leftarrow_{\$} \mathcal{K}$, so the challenge bit b_{Ind} is computationally hidden from $\mathcal{A}$. So in the following analysis, we take it for granted that $\mathsf{Win}_{\mathsf{Auth}}$ does not happen.

Clearly, if π_U^i is not accepted, we have $k_U^i = \emptyset$ and $\mathsf{Test}(U, i)$ will return $\perp$ directly regardless of the value of b_{Ind}. Without loss of generality, we assume that π_U^i is accepted when $\mathcal{A}$ makes $\mathsf{Test}(U, i)$ queries.

To analyze the distributions of $k_0 = k_U^i$, we consider the following five cases.

- **Case 1**: $U = \mathsf{S}$, and C is corrupted when π_{S}^{i} is accepted. According to **TA2** (cf. Table 2), $\mathcal{A}$ is not allowed to make Test queries on such $(U = \mathsf{S}, i)$.
- **Case 2**: $U = \mathsf{S}$, and C is *not* corrupted when π_{S}^{i} is accepted. This is in fact the conditions (1)$\wedge$(2.1) in the definition of $\mathsf{Win}_{\mathsf{Auth}}$ (cf. Def. 4). Since $\mathsf{Win}_{\mathsf{Auth}}$ does not occur, it implies that (3) does not occur, i.e., π_{S}^{i} is partnered to a unique oracle π_{C}^{j}. This means that the messages sent from π_{S}^{i} are received

by π_{C}^j, and the messages sent from π_{C}^j are received by π_{S}^i as well. So $\mathcal{A}$ does not interfere with the interactions between π_{S}^i and π_{C}^j, and in particular, does not interfere with the execution of $\mathsf{aPAKE.Protocol}$ between π_{S}^i and π_{C}^j. There are two cases when C is not corrupted. If $\mathcal{A}$ has not queried $\mathsf{CorruptPW}$, then according to the security of aPAKE, the aPAKE session key $k_{\mathsf{S}}^{\mathsf{aPAKE}}$ established by π_{S}^i is pseudorandom to $\mathcal{A}$. Otherwise, $\mathcal{A}$ has not queried $\mathsf{CorruptBM}$, then according to the security of SS, the sampling w of W has average conditional entropy of at least $\tilde{m} = \omega(\log \lambda)$ conditioned on the sketch s, so $\mathcal{A}$ can hardly make a random oracle query on (w, x), and the value of $d = \mathsf{H_d}(w, x)$ is uniformly random from $\mathcal{A}$'s view. In either case, $\mathcal{A}$ can hardly make a random oracle query on $k_{\mathsf{S}}^{\mathsf{aPAKE}} \| d$, and thus the value of $k_0 = k_{\mathsf{S}}^i = \mathsf{H_{key}}(k_{\mathsf{S}}^{\mathsf{aPAKE}} \| d)$ is uniformly random from $\mathcal{A}$'s view and is indistinguishable from $k_1 \leftarrow_{\$} \mathcal{K}$.

- **Case 3**: $U = \mathsf{C}$, and S is corrupted when π_{C}^i is accepted. According to **TA1** (cf. Table 2), $\mathcal{A}$ is not allowed to make Test queries on such $(U = \mathsf{C}, i)$.
- **Case 4**: $U = \mathsf{C}$, and C is corrupted when π_{C}^i is accepted. According to **TA3** (cf. Table 2), $\mathcal{A}$ is not allowed to make Test queries on such $(U = \mathsf{C}, i)$.
- **Case 5**: $U = \mathsf{C}$, and both S and C are *not* corrupted when π_{C}^i is accepted. This is in fact the conditions (1) $\wedge$ (2.2) in the definition of $\mathsf{Win_{Auth}}$ (cf. Def. 4). Since $\mathsf{Win_{Auth}}$ does not occur, it implies that (3) does not occur, i.e., π_{C}^i is partnered to a unique oracle π_{S}^j. The following analysis is the same as that for Case 2. Namely, it implies that $\mathcal{A}$ does not interfere with the execution of $\mathsf{aPAKE.Protocol}$ between π_{C}^i and π_{S}^j. Similarly, there are two cases when C is not corrupted. If $\mathcal{A}$ has not queried $\mathsf{CorruptPW}$, then according to the security of aPAKE, the aPAKE session key $k_{\mathsf{C}}^{\mathsf{aPAKE}}$ established by π_{C}^i is pseudorandom to $\mathcal{A}$. Otherwise, $\mathcal{A}$ has not queried $\mathsf{CorruptBM}$, then according to the security of SS, the sampling w of W has average conditional entropy of at least $\tilde{m} = \omega(\log \lambda)$ conditioned on the sketch s, so $\mathcal{A}$ can hardly make a random oracle query on (w, x), and the value of $d = \mathsf{H_d}(w, x)$ is uniformly random from $\mathcal{A}$'s view. In either case, $\mathcal{A}$ can hardly make a random oracle query on $k_{\mathsf{C}}^{\mathsf{aPAKE}} \| d$, and thus the value of $k_0 = k_{\mathsf{C}}^i = \mathsf{H_{key}}(k_{\mathsf{C}}^{\mathsf{aPAKE}} \| d)$ is uniformly random from $\mathcal{A}$'s view and is indistinguishable from $k_1 \leftarrow_{\$} \mathcal{K}$.

Overall, for **Cases 1, 3, 4**, $\mathcal{A}$ is not allowed to make Test queries on such (U, i), while for **Cases 2, 5**, the real session keys $k_0 = k_U^i$ are computationally indistinguishable from $k_1 \leftarrow_{\$} \mathcal{K}$, and thus the challenge bit b_{Ind} is computationally hidden in these Test queries when $\mathsf{Win_{Auth}}$ does not happen. This shows that $\mathsf{Adv}^{\mathsf{Ind}}_{\mathsf{2fAKE},\ell,\mathcal{A}} = \left|\Pr[\mathsf{Win_{Ind}}] - \frac{1}{2}\right| \leq \mathsf{negl}(\lambda) + \Pr[\mathsf{Win_{Auth}}] \leq \mathsf{negl}(\lambda) + \frac{Q}{|\mathcal{D}_{\mathsf{pw}}|}$, and the indistinguishability of our $\mathsf{2fAKE}$ follows. □

5.3 Proof of Zero-Knowledge

To prove the zero-knowledge of our $\mathsf{2fAKE}$, we need to construct a simulator $\mathsf{Sim} = (\mathsf{Sim.Enroll}, \{\mathsf{Sim}.\pi_{\mathsf{C}}^i\}_{i \in [\ell]})$, such that without having the biometric W of C, $\mathsf{Sim.Enroll}(\mathsf{pw})$ can generate simulated $(\mathsf{edata}', \mathsf{pub}')$ and $\{\mathsf{Sim}.\pi_{\mathsf{C}}^i(\mathsf{pw}, \mathsf{pub}')\}_{i \in [\ell]}$

can generate simulated transcripts of the protocol instances for C, which are indistinguishable from honestly generated ones. The description of $\mathsf{Sim} = (\mathsf{Sim.Enroll}, \{\mathsf{Sim}.\pi^i_{\mathsf{C}}\}_{i\in[\ell]})$ is as follows.

- Firstly, Sim maintains two lists $\mathsf{HList}_{\mathsf{sig}}, \mathsf{HList}_{\mathsf{d}}$ for random oracle queries of $\mathsf{H}_{\mathsf{sig}}$ and H_{d}, respectively.
- **Description of** $\mathsf{Sim.Enroll}(\mathsf{pw})$**:** To generate simulated $(\mathsf{edata}', \mathsf{pub}')$ without using the biometric W, $\mathsf{Sim.Enroll}$ samples ssk directly from the signing key space, i.e., $ssk \leftarrow_{\$} \mathcal{SK}$, picks $x \leftarrow_{\$} \{0,1\}^\lambda$ uniformly, and records $(*, x, ssk)$ in $\mathsf{HList}_{\mathsf{sig}}$, which implicitly sets $\mathsf{H}_{\mathsf{sig}}(w, x) := ssk$ for a sampling w of W. Sim also records ssk and will directly retrieve it when needed rather than regenerate it every time. Similarly, Sim picks $d \leftarrow_{\$} \{0,1\}^\lambda$ uniformly, records $(*, x, d)$ in $\mathsf{HList}_{\mathsf{d}}$, which implicitly sets $\mathsf{H}_{\mathsf{d}}(w, x) := d$, and records d for retrieving it in the future. Moreover, $\mathsf{Sim.Enroll}$ invokes $vk := \mathsf{VK.Gen}(ssk)$ and $\mathsf{rw} \leftarrow \mathsf{aPAKE.Enroll}(\mathsf{pw})$ honestly, the same as the $\mathsf{2fAKE.Enroll}$ algorithm. Then $\mathsf{Sim.Enroll}$ simulates the sketch s by invoking the simulator $\mathsf{SS.Sim}$ supported by SS (cf. Def. 6 in Appendix A.1). Finally, $\mathsf{Sim.Enroll}$ sets $\mathsf{edata}' := (vk, \mathsf{rw}, d)$ and $\mathsf{pub}' := (s, x)$.

  Clearly, if $\mathcal{A}$ never queries $\mathsf{H}_{\mathsf{sig}}$ and H_{d} on (w, x), the simulations of $ssk \leftarrow_{\$} \mathcal{SK}$ and $d \leftarrow_{\$} \{0,1\}^\lambda$ are perfect from $\mathcal{A}$'s view, and thus the distributions of $\mathsf{edata}' = (vk, \mathsf{rw}, d)$ and $\mathsf{pub}' = (s, x)$ are indistinguishable from the honestly generated edata and pub.

  On the other hand, since $\mathcal{A}$ does not obtain a sampling w of W,[4] $\mathcal{A}$ can hardly query $\mathsf{H}_{\mathsf{sig}}$ or H_{d} on (w, x). This is because the only information about w that $\mathcal{A}$ may obtain is the sketch s, but conditioned on the sketch s, w has average conditional entropy of at least $\tilde{m} = \omega(\log \lambda)$, according to the security of SS.

  Combining the above arguments together, the $(\mathsf{edata}', \mathsf{pub}')$ simulated by $\mathsf{Sim.\ Enroll}(\mathsf{pw})$ is indistinguishable from the $(\mathsf{edata}, \mathsf{pub})$ generated by $\mathsf{2fAKE.Enroll}(W, \mathsf{pw})$.
- **Description of** $\{\mathsf{Sim}.\pi^i_{\mathsf{C}}(\mathsf{pw}, \mathsf{pub}')\}_{i\in[\ell]}$**:** To simulate the transcripts of the protocol instances for C without using the biometric W, $\mathsf{Sim}.\pi^i_{\mathsf{C}}$ firstly uses pw to execute the aPAKE protocol honestly with $\mathsf{S}/\mathcal{A}$ to get $(\Psi^{\mathsf{aPAKE}}_{\mathsf{C}}, k^{\mathsf{aPAKE}}_{\mathsf{C}})$, the same as the $\mathsf{2fAKE.Protocol}$. If $\Psi^{\mathsf{aPAKE}}_{\mathsf{C}} \neq \mathbf{accept}$, $\mathsf{Sim}.\pi^i_{\mathsf{C}}$ returns $(\Psi_{\mathsf{C}} := \mathbf{reject}, k_{\mathsf{C}} := \emptyset)$. Otherwise, $\mathsf{Sim}.\pi^i_{\mathsf{C}}$ receives t_{S} from with $\mathsf{S}/\mathcal{A}$, and will not regenerate d from the biometric W, but retrieves the d recorded by $\mathsf{Sim.Enroll}$ directly, and checks whether $t_{\mathsf{S}} = \mathsf{H}_{\mathsf{S}}(k^{\mathsf{aPAKE}}_{\mathsf{C}} \| d)$ holds. If $t_{\mathsf{S}} \neq \mathsf{H}_{\mathsf{S}}(k^{\mathsf{aPAKE}}_{\mathsf{C}} \| d)$, $\mathsf{Sim}.\pi^i_{\mathsf{C}}$ returns $(\Psi_{\mathsf{C}} := \mathbf{reject}, k_{\mathsf{C}} := \emptyset)$ as well. Otherwise, $\mathsf{Sim}.\pi^i_{\mathsf{C}}$ will also not regenerate ssk from the biometric W, but retrieve the ssk recorded by $\mathsf{Sim.Enroll}$ directly. With ssk, $\mathsf{Sim}.\pi^i_{\mathsf{C}}$ then generates a signature $\sigma \leftarrow \mathsf{SIG.Sign}(ssk, \mathsf{trans} \| t_{\mathsf{S}})$. Finally, $\mathsf{Sim}.\pi^i_{\mathsf{C}}$ computes $t_{\mathsf{C}} := \mathsf{H}_{\mathsf{C}}(k^{\mathsf{aPAKE}}_{\mathsf{C}} \| d)$, returns $(\Psi_{\mathsf{C}} := \mathbf{accept}, k_{\mathsf{C}} := \mathsf{H}_{\mathsf{key}}(k^{\mathsf{aPAKE}}_{\mathsf{C}} \| d))$, and sends (σ, t_{C}) to $\mathsf{S}/\mathcal{A}$.

[4] This is due to our formalization of zero-knowledge in Def. 4 and is reasonable. Otherwise, if $\mathcal{A}$ already obtains a sampling w of W (e.g., by corrupting client), it is meaningless to consider zero-knowledge of W.

Clearly, there are only two differences between $\mathsf{Sim}.\pi^i_{\mathsf{C}}$ and the real $\mathsf{2fAKE.Protocol}$. The first one is that $\mathsf{Sim}.\pi^i_{\mathsf{C}}$ uses the ssk recorded by $\mathsf{Sim.Enroll}$ to generate the signature σ, while the client C in $\mathsf{2fAKE.Protocol}$ regenerates ssk from a sampling w' of the biometric W. The second one is that $\mathsf{Sim}.\pi^i_{\mathsf{C}}$ uses the d recorded by $\mathsf{Sim.Enroll}$ to check the validity of t_{S} instead of regenerating it from a sampling w' of the biometric W. According to the correctness of SS, as long as the two samplings w' and w are within distance at most t, the ssk and d regenerated from w' are the same as the ssk and d generated by $\mathsf{Sim.Enroll}$. Consequently, the transcripts produced by $\mathsf{Sim}.\pi^i_{\mathsf{C}}$ are identical to the transcripts honestly generated by C in $\mathsf{2fAKE.Protocol}$.

Overall, the $(\mathsf{edata}', \mathsf{pub}')$ produced by $\mathsf{Sim.Enroll}(\mathsf{pw})$ and the transcripts simulated by $\{\mathsf{Sim}.\pi^i_{\mathsf{C}}(\mathsf{pw}, \mathsf{pub}')\}_{i\in[\ell]}$ are indistinguishable from those honestly generated ones, and thus the challenge bit b_{ZK} is computationally hidden in the security experiment $\mathsf{Exp}^{\mathsf{ZK}}_{\mathsf{2fAKE},\ell,\mathcal{A},\mathsf{Sim}}$. This shows that $\mathsf{Adv}^{\mathsf{ZK}}_{\mathsf{2fAKE},\ell,\mathcal{A},\mathsf{Sim}} = |\Pr[\mathsf{Win}_{\mathsf{ZK}}] - \frac{1}{2}| \leq \mathsf{negl}(\lambda)$, and the zero-knowledge of our $\mathsf{2fAKE}$ follows. □

6 Instantiation from Post-Quantum Assumptions

We show how to instantiate our generic construction to obtain $\mathsf{2fAKE}$ protocol from post-quantum assumptions.

- *Instantiations of* SS. Secure sketch is an information-theoretical primitive. The popular candidates are error-correction code-based SS [12, 27, 42, 43].
- *Instantiations of* SIG. There are many secure digital signature schemes, and any instantiation with canonical key generation works. To obtain instantiations from post-quantum assumptions, we can use lattice-based signatures in [17, 36] or CRYSTALS-Dilithium, Falcon in NIST Post-Quantum Cryptography (PQC) finalists [11, 14], or isogeny-based ones [6].
- *Instantiations of* aPAKE. In [16], Gentry et al. propose a generic construction of aPAKE protocol from digital signature and PAKE in the random oracle. By instantiating the underlying digital signature and PAKE with post-quantum secure ones like [17, 36, 11, 14, 6] and [39, 1], respectively, we immediately obtain concrete aPAKE protocol from post-quantum assumptions.

Efficiency Analysis. Next we analyze the concrete efficiency of our $\mathsf{2fAKE}$ protocol. For concreteness, we instantiate our $\mathsf{2fAKE}$ protocol with the secure sketch in [12], the digital signature Falcon [14], and the aPAKE protocol in [16] which is in turn instantiated with Falcon [14] and the PAKE protocol CAKE in [1]. For completeness, we illustrate the resulting $\mathsf{2fAKE}$ instantiation in Fig. 4 in Appendix B. Furthermore, we implement our instantiation on Ubuntu 20.04 with two Intel(R) Core(TM) i7-11800H CPUs and 4 GB memory, where we use PQClean [28] for Falcon.

(1) Storage size: Our experiments show that the enrollment data $\mathsf{edata} = (vk, \mathsf{rw}, d)$ stored by the server is just 3.1 KB for one client.

(2) Rounds: According to Fig. 3, our 2fAKE adds two extra rounds, t_S and (t_C, σ), to the underlying aPAKE protocol. Fortunately, there are two cases in which our 2fAKE adds only one extra round to aPAKE, or even adds no round to aPAKE at all.

- If the last round of aPAKE is sent from server to client, we can merge t_S of our 2fAKE with it, and hence only one additional round is needed for our 2fAKE.
- If the last round of aPAKE is sent from client to server, and moreover, the server can compute aPAKE session key before the second last round, then we can merge t_S with the second last round of aPAKE and merge (t_C, σ) with the last round of aPAKE. In this case, no additional round is needed for our 2fAKE.

Indeed, the aPAKE protocol in [16] falls into the second case, and then our 2fAKE protocol has the same number of rounds as the aPAKE protocol in [16], i.e., 3 rounds. See Fig. 4 in Appendix B for our 2fAKE instantiation.

(3) Communication Complexity: By Fig. 3, our 2fAKE adds two hash values t_S, t_C and one signature σ to the communication of the underlying aPAKE. Note that t_C is designed for explicit authentication of the client's password, and thus we can eliminate it if the underlying aPAKE has already provided explicit authentication for the client's password, i.e., the server will abort if the client does not have the correct pw. This is achieved by many aPAKE protocols including the one in [16]. Consequently, the communication of our 2fAKE costs just 4.3 KB.

(4) Computation Complexity: Our experiments show that the running time of the client is about 25 ms and the running time of the server is about 0.2 ms. Here the client spends more time than the server, since the recovery algorithm of the secure sketch takes up about half of the running time of the client. We leave a more efficient instantiation as an interesting future work.

References

[1] Beguinet, H., Chevalier, C., Pointcheval, D., Ricosset, T., Rossi, M.: GeT a CAKE: Generic transformations from key encaspulation mechanisms to password authenticated key exchanges. In: Tibouchi, M., Wang, X. (eds.) ACNS 23, Part II. LNCS, vol. 13906, pp. 516–538. Springer, Heidelberg (Jun 2023). https://doi.org/10.1007/978-3-031-33491-7_19

[2] Bellare, M., Pointcheval, D., Rogaway, P.: Authenticated key exchange secure against dictionary attacks. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 139–155. Springer, Heidelberg (May 2000). https://doi.org/10.1007/3-540-45539-6_11

[3] Bellare, M., Rogaway, P.: Entity authentication and key distribution. In: Stinson, D.R. (ed.) CRYPTO'93. LNCS, vol. 773, pp. 232–249. Springer, Heidelberg (Aug 1994). https://doi.org/10.1007/3-540-48329-2_21

[4] Bellovin, S.M., Merritt, M.: Encrypted key exchange: Password-based protocols secure against dictionary attacks. In: 1992 IEEE Symposium on Security and Privacy. pp. 72–84. IEEE Computer Society Press (May 1992). https://doi.org/10.1109/RISP.1992.213269

[5] Bellovin, S.M., Merritt, M.: Augmented encrypted key exchange: A password-based protocol secure against dictionary attacks and password file compromise. In: Denning, D.E., Pyle, R., Ganesan, R., Sandhu, R.S., Ashby, V. (eds.) ACM CCS 93. pp. 244–250. ACM Press (Nov 1993). https://doi.org/10.1145/168588.168618
[6] Beullens, W., Kleinjung, T., Vercauteren, F.: CSI-FiSh: Efficient isogeny based signatures through class group computations. In: Galbraith, S.D., Moriai, S. (eds.) ASIACRYPT 2019, Part I. LNCS, vol. 11921, pp. 227–247. Springer, Heidelberg (Dec 2019). https://doi.org/10.1007/978-3-030-34578-5_9
[7] Canetti, R., Krawczyk, H.: Security analysis of IKE's signature-based key-exchange protocol. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 143–161. Springer, Heidelberg (Aug 2002). https://doi.org/10.1007/3-540-45708-9_10, https://eprint.iacr.org/2002/120/
[8] Canetti, R., Krawczyk, H.: Universally composable notions of key exchange and secure channels. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 337–351. Springer, Heidelberg (Apr / May 2002). https://doi.org/10.1007/3-540-46035-7_22
[9] Daugman, J.: Understanding biometric entropy and iris capacity: Avoiding identity collisions on national scales. Adv. Artif. Intell. Mach. Learn. **4**(2), 2152–2163 (2024). https://doi.org/10.54364/AAIML.2024.42123
[10] Denis, F.: libsodium, https://github.com/jedisct1/libsodium
[11] Dilithium: NIST post-quantum cryptography standardization, https://pq-crystals.org/dilithium/
[12] Dodis, Y., Ostrovsky, R., Reyzin, L., Smith, A.D.: Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. SIAM J. Comput. **38**(1), 97–139 (2008). https://doi.org/10.1137/060651380
[13] El-Tokhy, M.S.: Robust multimodal biometric authentication algorithms using fingerprint, iris and voice features fusion. J. Intell. Fuzzy Syst. **40**(1), 647–672 (2021)
[14] Falcon: NIST post-quantum cryptography standardization, https://falcon-sign.info/
[15] Fei, S., Yan, Z., Ding, W., Xie, H.: Security vulnerabilities of SGX and countermeasures: A survey. ACM Comput. Surv. **54**(6), 126:1–126:36 (2022). https://doi.org/10.1145/3456631, https://doi.org/10.1145/3456631
[16] Gentry, C., MacKenzie, P., Ramzan, Z.: A method for making password-based key exchange resilient to server compromise. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 142–159. Springer, Heidelberg (Aug 2006). https://doi.org/10.1007/11818175_9
[17] Gentry, C., Peikert, C., Vaikuntanathan, V.: Trapdoors for hard lattices and new cryptographic constructions. In: Ladner, R.E., Dwork, C. (eds.) 40th ACM STOC. pp. 197–206. ACM Press (May 2008). https://doi.org/10.1145/1374376.1374407
[18] Gielczyk, A., Choras, M.: Intelligent human-centred mobile authentication system based on palmprints. J. Intell. Fuzzy Syst. **39**(6), 8217–8224 (2020)
[19] Günther, C.G.: An identity-based key-exchange protocol. In: Quisquater, J.J., Vandewalle, J. (eds.) EUROCRYPT'89. LNCS, vol. 434, pp. 29–37. Springer, Heidelberg (Apr 1990). https://doi.org/10.1007/3-540-46885-4_5
[20] Han, Y., Xu, C., Jiang, C., Chen, K.: A secure two-factor authentication key exchange scheme. IEEE Transactions on Dependable and Secure Computing pp. 1–13 (2024). https://doi.org/10.1109/TDSC.2024.3382359

[21] Jarecki, S., Jubur, M., Krawczyk, H., Saxena, N., Shirvanian, M.: Two-factor password-authenticated key exchange with end-to-end security. ACM Trans. Priv. Secur. **24**(3) (Apr 2021). https://doi.org/10.1145/3446807

[22] Jarecki, S., Krawczyk, H., Shirvanian, M., Saxena, N.: Two-factor authentication with end-to-end password security. In: Abdalla, M., Dahab, R. (eds.) PKC 2018, Part II. LNCS, vol. 10770, pp. 431–461. Springer, Heidelberg (Mar 2018). https://doi.org/10.1007/978-3-319-76581-5_15

[23] Jarecki, S., Krawczyk, H., Xu, J.: OPAQUE: An asymmetric PAKE protocol secure against pre-computation attacks. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part III. LNCS, vol. 10822, pp. 456–486. Springer, Heidelberg (Apr / May 2018). https://doi.org/10.1007/978-3-319-78372-7_15

[24] Jiang, C., Xu, C., Han, Y., Zhang, Z., Chen, K.: Two-factor authenticated key exchange from biometrics with low entropy rates. IEEE Transactions on Information Forensics and Security pp. 1–1 (2024). https://doi.org/10.1109/TIFS.2024.3372812

[25] Jiang, M., Liu, S., Han, S., Gu, D.: Fuzzy authenticated key exchange with tight security. In: Atluri, V., Di Pietro, R., Jensen, C.D., Meng, W. (eds.) ESORICS 2022, Part II. LNCS, vol. 13555, pp. 337–360. Springer, Heidelberg (Sep 2022). https://doi.org/10.1007/978-3-031-17146-8_17

[26] Jiang, P., Wang, Q., Lin, X., Zhou, M., Ding, W., Wang, C., Shen, C., Li, Q.: Securing liveness detection for voice authentication via pop noises. IEEE Trans. Dependable Secur. Comput. **20**(2), 1702–1718 (2023)

[27] Juels, A., Sudan, M.: A fuzzy vault scheme. Des. Codes Cryptogr. **38**(2), 237–257 (2006). https://doi.org/10.1007/S10623-005-6343-Z

[28] Kannwischer, M.J., Schwabe, P., Stebila, D., Wiggers, T.: Improving software quality in cryptography standardization projects. In: IEEE European Symposium on Security and Privacy, EuroS&P 2022. pp. 19–30. IEEE Computer Society (2022). https://doi.org/10.1109/EuroSPW55150.2022.00010

[29] Katz, J., Lindell, Y.: Introduction to Modern Cryptography, Second Edition. 2nd edn. (2014)

[30] Katz, J., Ostrovsky, R., Yung, M.: Efficient password-authenticated key exchange using human-memorable passwords. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 475–494. Springer, Heidelberg (May 2001). https://doi.org/10.1007/3-540-44987-6_29

[31] Krawczyk, H.: HMQV: A high-performance secure Diffie-Hellman protocol. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 546–566. Springer, Heidelberg (Aug 2005). https://doi.org/10.1007/11535218_33

[32] Kyber: NIST post-quantum cryptography standardization, submission package for round 3, https://pq-crystals.org/kyber/

[33] LaMacchia, B.A., Lauter, K., Mityagin, A.: Stronger security of authenticated key exchange. In: Susilo, W., Liu, J.K., Mu, Y. (eds.) ProvSec 2007. LNCS, vol. 4784, pp. 1–16. Springer, Heidelberg (Nov 2007)

[34] Lei, J., Pei, Q., Wang, Y., Sun, W., Liu, X.: Privface: Fast privacy-preserving face authentication with revocable and reusable biometric credentials. IEEE Trans. Dependable Secur. Comput. **19**(5), 3101–3112 (2022)

[35] Li, Y., Schäge, S.: No-match attacks and robust partnering definitions: Defining trivial attacks for security protocols is not trivial. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017. pp. 1343–1360. ACM Press (Oct / Nov 2017). https://doi.org/10.1145/3133956.3134006

[36] Lyubashevsky, V.: Lattice signatures without trapdoors. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 738–755. Springer, Heidelberg (Apr 2012). https://doi.org/10.1007/978-3-642-29011-4_43
[37] NIST: NIST post-quantum cryptography standardization, https://csrc.nist.gov/projects/post-quantum-cryptography/
[38] Pointcheval, D., Zimmer, S.: Multi-factor authenticated key exchange. In: Bellovin, S.M., Gennaro, R., Keromytis, A.D., Yung, M. (eds.) ACNS 08. LNCS, vol. 5037, pp. 277–295. Springer, Heidelberg (Jun 2008). https://doi.org/10.1007/978-3-540-68914-0_17
[39] Santos, B.F.D., Gu, Y., Jarecki, S.: Randomized half-ideal cipher on groups with applications to UC (a)PAKE. In: Hazay, C., Stam, M. (eds.) EUROCRYPT 2023, Part V. LNCS, vol. 14008, pp. 128–156. Springer, Heidelberg (Apr 2023). https://doi.org/10.1007/978-3-031-30589-4_5
[40] Shoup, V.: Security analysis of itSPAKE2+. In: Pass, R., Pietrzak, K. (eds.) TCC 2020, Part III. LNCS, vol. 12552, pp. 31–60. Springer, Heidelberg (Nov 2020). https://doi.org/10.1007/978-3-030-64381-2_2
[41] Wang, M., He, K., Chen, J., Li, Z., Zhao, W., Du, R.: Biometrics-authenticated key exchange for secure messaging. In: Vigna, G., Shi, E. (eds.) ACM CCS 2021. pp. 2618–2631. ACM Press (Nov 2021). https://doi.org/10.1145/3460120.3484746
[42] Wen, Y., Liu, S.: Robustly reusable fuzzy extractor from standard assumptions. In: Peyrin, T., Galbraith, S. (eds.) ASIACRYPT 2018, Part III. LNCS, vol. 11274, pp. 459–489. Springer, Heidelberg (Dec 2018). https://doi.org/10.1007/978-3-030-03332-3_17
[43] Woodage, J., Chatterjee, R., Dodis, Y., Juels, A., Ristenpart, T.: A new distribution-sensitive secure sketch and popularity-proportional hashing. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part III. LNCS, vol. 10403, pp. 682–710. Springer, Heidelberg (Aug 2017). https://doi.org/10.1007/978-3-319-63697-9_23
[44] Wu, C., He, K., Chen, J., Zhao, Z., Du, R.: Liveness is not enough: Enhancing fingerprint authentication with behavioral biometrics to defeat puppet attacks. In: Capkun, S., Roesner, F. (eds.) USENIX Security 2020. pp. 2219–2236. USENIX Association (Aug 2020)
[45] Xie, Q., Wong, D.S., Wang, G., Tan, X., Chen, K., Fang, L.: Provably secure dynamic id-based anonymous two-factor authenticated key exchange protocol with extended security model. IEEE Transactions on Information Forensics and Security **12**(6), 1382–1392 (2017). https://doi.org/10.1109/TIFS.2017.2659640
[46] Yankov, M.P., Olsen, M.A., Stegmann, M.B., Christensen, S.S., Forchhammer, S.: Fingerprint entropy and identification capacity estimation based on pixel-level generative modelling. IEEE Trans. Inf. Forensics Secur. **15**, 56–65 (2020). https://doi.org/10.1109/TIFS.2019.2916406

Appendix

A Additional Preliminaries

A.1 Secure Sketch

Secure sketch (SS) was proposed in [12], and it can produce a public value s (called sketch) that allows one to recover $w \in \mathcal{W}$ from any sufficiently close value $w' \in \mathcal{W}$, with the security requirement that s does not reveal too much information of w.

Before presenting the definition of secure sketch, we first define some notations. Let $\mathcal{W}$ be a metric space. For any $w_1, w_2 \in \mathcal{W}$, let $\mathsf{dis}(w_1, w_2)$ denote the distance between w_1 and w_2. For random variables X and Y, the min-entropy of X is defined as $\mathbf{H}_\infty(X) := -\log(\max_x \Pr[X = x])$, and the average min-entropy of X conditioned on Y is defined as $\widetilde{\mathbf{H}}_\infty(X|Y) := -\log(\mathbb{E}_{y \leftarrow_s Y}[\max_x \Pr[X = x|Y = y]])$, where $\mathbb{E}$ denotes the mathematical expectation.

Definition 5 (Secure Sketch). *An $(\mathcal{W}, m, \tilde{m}, t)$-secure sketch $\mathsf{SS} = (\mathsf{SS.Gen}, \mathsf{SS.Rec})$ for metric space $\mathcal{W}$ consists of two PPT algorithms:*

- *$s \leftarrow \mathsf{SS.Gen}(w)$: The sketch generation algorithm takes $w \in \mathcal{W}$ as input, and outputs a sketch s.*
- *$\hat{w} \leftarrow \mathsf{SS.Rec}(w', s)$: The recovering algorithm takes as input $w' \in \mathcal{W}$ and a sketch s, and outputs $\hat{w}$.*

Correctness. *For $\forall w, w' \in \mathcal{W}$ with $\mathsf{dis}(w, w') \leq t$ and $\forall s \leftarrow \mathsf{SS.Gen}(w)$, it holds that $\mathsf{SS.Rec}(w', s) = w$.*

Security. *For $\forall$ distribution W over $\mathcal{W}$ with $\mathbf{H}_\infty(W) \geq m$, it holds that $\widetilde{\mathbf{H}}_\infty(W|\mathsf{SS.Gen}(W)) \geq \tilde{m}$.*

Let $\mathbf{W}$ be a distribution of all possible distributions W over $\mathcal{W}$ (e.g., the distribution of all possible biometrics like faces, fingerprints and iris). We require that there is a uniform way to simulate the distribution of sketches $s \leftarrow \mathsf{SS.Gen}(w)$ for randomly chosen samplings $w \leftarrow W$ of randomly chosen distributions $W \leftarrow \mathbf{W}$, given only $\mathbf{W}$. Formally, we have the following definition.

Definition 6 (Simulatable Sketches). *A secure sketch $\mathsf{SS} = (\mathsf{SS.Gen}, \mathsf{SS.Rec})$ has simulatable sketches, if these exists a PPT simulator $\mathsf{SS.Sim}$ that can simulate the sketches w.r.t. $\mathbf{W}$ in an indistinguishable way, i.e., such that the following two distributions are (statistically) indistinguishable:*

$$\{\ W \leftarrow \mathbf{W},\ w \leftarrow W,\ s \leftarrow \mathsf{SS.Gen}(w)\ :\ s\ \} \quad \overset{s}{\approx} \quad \{\ s \leftarrow \mathsf{SS.Sim}(\mathbf{W})\ :\ s\ \}.$$

Actually, we can always have such a simulator $\mathsf{SS.Sim}(\mathbf{W})$, which just samples $W \leftarrow \mathbf{W}$, $w \leftarrow W$ and outputs $s \leftarrow \mathsf{SS.Gen}(w)$, so that Def. 6 is easily satisfied. For some concrete distributions $\mathbf{W}$, there might be simpler ways to simulate the sketches.

A.2 Digital Signature

We consider digital signature (SIG) with *canonical* key generation algorithm, as defined below.

Definition 7 (Digital Signature). *A digital signature (SIG) scheme* $\mathsf{SIG} = (\mathsf{SIG.Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ *with message space* $\mathcal{M}$ *consists of three PPT algorithms:*

- $(vk, ssk) \leftarrow \mathsf{SIG.Gen}$*: The key generation algorithm outputs a pair of verification key* vk *and signing key* ssk*.*
- $\sigma \leftarrow \mathsf{Sign}(ssk, m)$*: The signing algorithm takes as input a signing key* ssk *and a message* $m \in \mathcal{M}$*, and outputs a signature* σ*.*
- $0/1 \leftarrow \mathsf{Vrfy}(vk, m, \sigma)$*: The deterministic verification algorithm takes as input a verification key* vk*, a message* $m \in \mathcal{M}$ *and a signature* σ*, and outputs a bit indicating whether* σ *is a valid signature for* m *w.r.t.* vk*.*

Correctness. *For* $\forall (vk, ssk) \leftarrow \mathsf{SIG.Gen}$, $\forall m \in \mathcal{M}$ *and* $\forall \sigma \leftarrow \mathsf{Sign}(ssk, m)$*, it holds that* $\mathsf{Vrfy}(vk, m, \sigma) = 1$*.*

Security. *We consider the standard security of existential unforgeability against chosen message attacks (EUF-CMA) for* SIG*. More precisely, for any PPT adversary* $\mathcal{A}$*, it holds that*

$$\mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{SIG},\mathcal{A}}(\lambda) := \Pr\left[\begin{array}{c}(vk, ssk) \leftarrow_{\$} \mathsf{SIG.Gen},\\ (m^*, \sigma^*) \leftarrow_{\$} \mathcal{A}^{\mathcal{O}_{\mathrm{SIGN}}(\cdot)}(vk)\end{array} : \begin{array}{c}m^* \notin \mathcal{Q}_{\mathsf{sig}} \wedge\\ \mathsf{Vrfy}(vk, m^*, \sigma^*) = 1\end{array}\right] \leq \mathsf{negl}(\lambda),$$

where the oracle $\mathcal{O}_{\mathrm{SIGN}}(m)$ *computes* $\sigma \leftarrow_{\$} \mathsf{Sign}(ssk, m)$ *and returns* σ *to* $\mathcal{A}$*, and the set* $\mathcal{Q}_{\mathsf{sig}}$ *consists of all messages* m *that* $\mathcal{A}$ *queries to* $\mathcal{O}_{\mathrm{SIGN}}(\cdot)$*.*

Canonical Key Generation. *We require the key generation algorithm* $\mathsf{SIG.Gen}$ *to be canonical, in the sense that it generates* (vk, ssk) *by first sampling* ssk *uniformly at random from a signing key space* $\mathcal{SK}$ *and then computing* vk *deterministically from* ssk *with a verification key generation algorithm* $\mathsf{VK.Gen}$*, i.e.,* $ssk \leftarrow_{\$} \mathcal{SK}$ *and* $vk := \mathsf{VK.Gen}(ssk)$*.*

A.3 Augmented/Asymmetric PAKE (aPAKE)

Password-based Authenticated Key Exchange (PAKE) [4, 2, 30] allows two parties (client and server) to establish a session key via public channels if they share a low-entropy password pw in advance. However, this requires the server to store passwords of its clients, and if the server is compromised, the adversary can snatch the password pw from the server so as to impersonate the client.

To address this vulnerability, augmented/asymmetric PAKE (aPAKE) [5, 16] was proposed, where the server stores only a password file rw (usually a hash value $\mathsf{H}(\mathsf{pw})$) rather than the plain password pw of the client. A client can establish a session key with the server if the client has the pre-image of the password file.

Definition 8 (aPAKE). *An augmented/asymmetric password-based authenticated key exchange (aPAKE) protocol* $\mathsf{aPAKE} = (\mathsf{aPAKE.Setup}, \mathsf{aPAKE.Enroll}, \mathsf{aPAKE.Protocol})$ *consists of two PPT algorithms and a PPT interactive protocol.*

– $\mathsf{pp}_{\mathsf{aPAKE}} \leftarrow \mathsf{aPAKE.Setup}$*: The setup algorithm outputs public parameter* $\mathsf{pp}_{\mathsf{aPAKE}}$*, which serves as an implicit input of other algorithms.*
– $\mathsf{rw} \leftarrow \mathsf{aPAKE.Enroll}(\mathsf{pw})$*: The enrollment algorithm takes a password* pw *as input, and outputs a password file/enrollment data* rw*, which will be stored on a server.*
– $((\Psi_{\mathsf{C}}^{\mathsf{aPAKE}}, k_{\mathsf{C}}^{\mathsf{aPAKE}}), (\Psi_{\mathsf{S}}^{\mathsf{aPAKE}}, k_{\mathsf{S}}^{\mathsf{aPAKE}})) \leftarrow \mathsf{aPAKE.Protocol}(\mathsf{C}(\mathsf{pw}) \leftrightarrow \mathsf{S}(\mathsf{rw}))$*: The protocol is invoked by a client* C *and a server* S*, where* C *takes* pw *as input and* S *takes* rw *as input. After the execution of the protocol, the client* C *outputs a flag* $\Psi_{\mathsf{C}}^{\mathsf{aPAKE}} \in \{\emptyset, \textbf{\textit{accept}}, \textbf{\textit{reject}}\}$ *and a session key* $k_{\mathsf{C}}^{\mathsf{aPAKE}} \in \mathcal{K} \cup \{\emptyset\}$ *(where* $\mathcal{K}$ *denotes the session key space), and similarly, the server* S *outputs* $\Psi_{\mathsf{S}}^{\mathsf{aPAKE}}$ *and* $k_{\mathsf{S}}^{\mathsf{aPAKE}}$*. We use* trans *to denote the concatenation of messages transferred between* C *and* S *during the execution of the protocol.*

Correctness. *If the client* C *and the server* S *execute* $\mathsf{aPAKE.Protocol}(\mathsf{C}(\mathsf{pw}) \leftrightarrow \mathsf{S}(\mathsf{rw}))$ *honestly, where* $\mathsf{rw} \leftarrow \mathsf{aPAKE.Enroll}(\mathsf{pw})$*, they will both accept the session and output a same session key, i.e.,* $\Psi_{\mathsf{C}}^{\mathsf{aPAKE}} = \Psi_{\mathsf{S}}^{\mathsf{aPAKE}} = \textbf{\textit{accept}}$ *and* $k_{\mathsf{C}}^{\mathsf{aPAKE}} = k_{\mathsf{S}}^{\mathsf{aPAKE}} \neq \emptyset$*.*

Security. *We consider the security of indistinguishability and authentication for* aPAKE*. More precisely, indistinguishability and authentication of* aPAKE *require that:*

- *Indistinguishability requires that the session keys* $\Psi_{\mathsf{C}}^{\mathsf{aPAKE}} = \Psi_{\mathsf{S}}^{\mathsf{aPAKE}}$ *generated by* C *and* S *are computationally indistinguishable from uniformly random keys.*
- *Authentication requires that* S *accepts a session only if* C *possesses its own password* pw*, and* C *accepts a session only if* S *has the corresponding* rw*. Moreover, for a PPT adversary* $\mathcal{A}$ *who compromises* S *and gets* rw*, it is computationally hard for* $\mathcal{A}$ *to use* rw *to impersonate* C *to make* S *accept a session, unless* $\mathcal{A}$ *recovers* pw *from* rw*.*

We refer to [16, 23] for more detailed definition of the security of aPAKE*.*

B Figure of Our 2fAKE Instantiation

In Fig. 4, we illustrate the instantiation of our 2fAKE from the aPAKE protocol in [16], which is in turn instantiated from the PAKE protocol CAKE in [1]. The instantiation additionally relies on three symmetric encryption schemes $(\mathsf{E}_0, \mathsf{D}_0)$, $(\mathsf{E}_1, \mathsf{D}_1)$, $(\mathsf{E}_2, \mathsf{D}_2)$, a key encapsulation mechanism (KEM) (KEM.Gen, KEM.Encaps, KEM.Decaps), and four hash functions $(\mathsf{H}_0, \mathsf{H}_1, \mathsf{H}_2, \mathsf{H}_3)$.

As analyzed in [1], the KEM can be instantiated with Kyber in NIST Post-Quantum Cryptography (PQC) finalists [32], and then we can get 2fAKE from post-quantum assumptions. For the three symmetric encryption schemes, we instantiate them with proper working modes of AES [32], and for the hash functions, we instantiate them with BLAKE2b [10].

For completeness, we present the formal definitions of symmetric encryption and KEM as follows.

Definition 9 (Symmetric Encryption). *A symmetric encryption scheme* (E, D) *with key space* $\mathcal{K}$ *and message space* $\mathcal{M}$ *consists of two PPT algorithms:*

- $Em \leftarrow \mathsf{E}(k, m)$*: Taking as input a symmetric key* $k \in \mathcal{K}$ *and a message* $m \in \mathcal{M}$*, the encryption algorithm outputs a ciphertext* Em*.*
- $m/\perp \leftarrow \mathsf{D}(k, Em)$*: Taking as input a symmetric key* $k \in \mathcal{K}$ *and a ciphertext* Em*, the deterministic decryption algorithm outputs either a message* $m \in \mathcal{M}$ *or a special symbol* $\perp$ *indicating the failure of decryption.*

Correctness. *For* $\forall k \in \mathcal{K}$*,* $\forall m \in \mathcal{M}$ *and* $\forall Em \leftarrow \mathsf{E}(k, m)$*, it holds that* $\mathsf{D}(k, Em) = m$*.*

Definition 10 (KEM). *A key encapsulation mechanism (KEM) scheme* (KEM.Gen, KEM.Encaps, KEM.Decaps) *consists of three PPT algorithms:*

- $(pk, sk) \leftarrow \mathsf{KEM.Gen}$*: The key generation algorithm outputs a pair of public key* pk *and secret key* sk*.*
- $(c, K) \leftarrow \mathsf{KEM.Encaps}(pk)$*: Taking a public key* pk *as input, the encapsulation algorithm outputs a ciphertext* c *and an encapsulated key* K*.*
- $K/\perp \leftarrow \mathsf{KEM.Decaps}(sk, c)$*: Taking as input a secret key* sk *and a ciphertext* c*, the deterministic decapsulation algorithm outputs either an encapsulated key* K *or a special symbol* $\perp$ *indicating the failure of decapsulation.*

Correctness. *For* $\forall (pk, sk) \leftarrow \mathsf{KEM.Gen}$ *and* $\forall (c, K) \leftarrow \mathsf{KEM.Encaps}(pk)$*, it holds that* $\mathsf{KEM.Decaps}(sk, c) = K$*.*

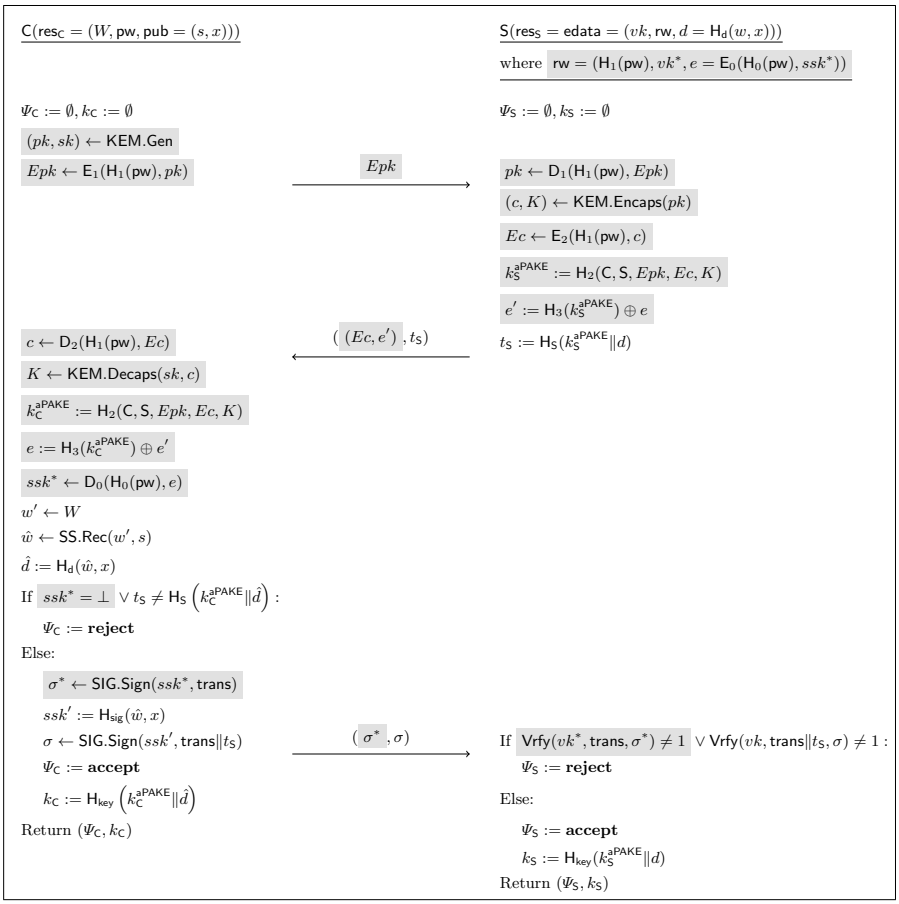

Fig. 4. Instantiation of our 2fAKE from the aPAKE protocol in [16], which is in turn instantiated from the PAKE protocol CAKE in [1]. Here the parts in gray boxes arise from the underlying (a)PAKE protocol [16, 1], where $(\mathsf{E}_0, \mathsf{D}_0)$, $(\mathsf{E}_1, \mathsf{D}_1)$, $(\mathsf{E}_2, \mathsf{D}_2)$ are symmetric encryption schemes, $(\mathsf{KEM.Gen}, \mathsf{KEM.Encaps}, \mathsf{KEM.Decaps})$ are algorithms of a key encapsulation mechanism (KEM), and $(\mathsf{H}_0, \mathsf{H}_1, \mathsf{H}_2, \mathsf{H}_3)$ are hash functions.