Two Efficient and Provably Secure Schemes for Server-Assisted Threshold Signatures

Shouhuai Xu[1]* and Ravi Sandhu[2]

[1] Department of Information and Computer Science
University of California at Irvine
shxu@ics.uci.edu
[2] SingleSignOn.Net and
Laboratory for Information Security Technology, George Mason University
sandhu@gmu.edu

Abstract. Secrecy of private signing keys is one of the most important issues in secure electronic commerce. A promising solution to this problem is to distribute the signing function among multiple parties. However, a threshold signature scheme typically assumes that the shared signing function can only be activated by a quorum number of parties, which is inappropriate in settings where a user employs some public servers for a threshold protection of her private signing function (therefore the name "server-assisted threshold signatures").

In this paper we present two efficient and provably secure schemes for server-assisted threshold signatures, where the signing function is activated by a user (but in certain enhanced way). The first one (we call `TPAKE-HTSig`) is tailored for the setting where a user has a networked device that is powerful enough to efficiently compute modular exponentiations. The second one (we call `LW-TSig`) is tailored for the setting where a user has a smart card without a cryptographic co-processor. Modular construction of the schemes ensures that any module can be substituted without weakening security of the resultant scheme, as long as the substitutive one satisfies certain security requirement. In addition to the two schemes, we also present a taxonomy of systems protecting private signing functions.

1 Introduction

Secrecy of private signing keys is one of the most important issues in secure electronic commerce. Threshold signatures enable us to achieve better security (i.e., compromise of certain number of participants does not expose the private key) and availability (i.e., a quorum number of participants suffice to sign messages). However, threshold signature schemes typically assume that the shared signing function can only be activated by a quorum number of participants; otherwise, the participant able to activate the function can always get signatures

* Work mostly done while at the Laboratory for Information Security Technology, George Mason University.

without compromising the private key. This assumption is relevant in certain settings (e.g., enterprise applications) but inappropriate in settings where a user employs some public servers (which perhaps provide service for economic incentives) to provide a threshold protection for her private signing function (therefore the name "server-assisted threshold signatures").

1.1 Our Contributions

We propose two efficient and provably secure schemes for server-assisted threshold signatures, where the signing function is activated by a user (but in certain enhanced way). The schemes are obtained via a *modular composition* approach, thereby any module can be substituted without weakening security of the resultant scheme, as long as the substitutive one (with, for instance, better performance) satisfies certain security requirements. The first scheme (we call `TPAKE-HTSig`) is tailored for the setting where a user has a networked device that is powerful enough to efficiently compute modular exponentiations. The second scheme (we call `LW-TSig`) is tailored for the setting where a user has a smart card with no cryptographic co-processor. In addition to the two schemes, we also present a taxonomy of systems protecting private signing functions.

Remark. Having said that our schemes allow the substitution of component modules as long as certain security requirement is satisfied, we stress that our approach is less general than the approach of Canetti [10].

Outline. In Section 2 we present certain cryptographic preliminaries. In Section 3 we present our first scheme `TPAKE-HTSig`, of which a formal security analysis is given in Appendix B. In Section 4 we present our second scheme `LW-TSig`. In Section 5, we present a taxonomy of systems for protecting private signing functions (including our schemes). We conclude in section 6. Due to space limitation, we eliminate from this version many discussions (including security analysis of `LW-TSig`), which will be found in the full version of this paper.

2 Cryptographic Preliminaries

Message Authentication Codes (MACs). A function family $\{f_k\}_k$, where $k \in \{0,1\}^\kappa$ for some security parameter κ, is a secure MAC family if any adversary $\mathcal{A}$ of probabilistic polynomial-time succeeds in the following game only with negligible probability. First a random key $k \in \{0,1\}^\kappa$ is chosen; then $\mathcal{A}$ adaptively chooses messages $m_1, \cdots, m_n$ and receives the corresponding MAC values $f_k(m_1), \cdots, f_k(m_n)$. $\mathcal{A}$ succeeds if it manages to generate a pair $\langle m, tag\rangle$, where $m \neq m_1, \cdots, m_n$ but $tag = f_k(m)$. We refer the reader to [2] for details.

Signature Schemes. A signature scheme `Sig` consists of three algorithms (`Sig.Init`, `Sig.Sig`, `Sig.Ver`). Taken as input a security parameter κ, the probabilistic key generation algorithm `Sig.Init` returns a pair of public and private keys (Y, X). Taken as input a message m and the private key X, the signing algorithm `Sig.Sig` outputs a signature σ. Taken as input a tag σ, the public key

Y, and a message m, the verification algorithm `Sig.Ver` returns TRUE if σ is a valid signature, and FALSE otherwise. We require a signature scheme be secure against adaptive chosen-message attack [17, 36].

THRESHOLD SIGNATURE SCHEMES. A threshold signature scheme `TSig` consists of three algorithms (`TSig.Init`, `TSig.Sig`, `TSig.Ver`). We focus on a *subclass* of signature schemes such that their threshold version falls into the following framework of specification; the motivation is to make concise the security proof of the resultant scheme. (Nonetheless, this subclass covers many *practical* and popular signature schemes [31, 5, 33, 11].)

1. Taken as input a security parameter κ, the number of tolerated corrupt servers t, and the number of servers n, `TSig.Init` outputs a public key Y so that the corresponding private key X is shared among the n servers $\{S_1, \cdots, S_n\}$ using an appropriate secret sharing scheme [34, 13, 28]. Let $X \overset{(t+1,n)}{\longleftrightarrow} (X^{(1)}, \cdots, X^{(n)})$ denote such a secret sharing so that S_i holds $X^{(i)}$, where $1 \leq i \leq n$.
2. To sign a message m, `TSig.Sig` runs as follows: S_{i_j} generates a partial signature $\sigma^{(i_j)} = g_1(m, X^{(i_j)})$, then anyone can compute $\sigma = g_2(\sigma^{(i_1)}, \cdots, \sigma^{(i_w)})$ from w valid partial signatures, where $t+1 \leq w \leq n$, $1 \leq i_j \leq n$ for $1 \leq j \leq w$, g_1 and g_2 are certain algorithms that depend on `Sig`.
3. `TSig.Ver` is the same as `Sig.Ver`.

A threshold signature scheme `TSig` is secure if, (1) it is unforgeable under adaptive chosen-message attack, where unforgeability is captured by presenting a polynomial-time simulator that is able to emulate `TSig` by having oracle access to the underlying `Sig`, and (2) it is robust, which means that an adversary having corrupted t servers is still unable to prevent it from functioning. Concrete threshold signature schemes can be found in, for instance, [16, 18, 30, 35].

A special and useful case of threshold signature schemes is the so-called two-party signature schemes `2Sig`=(`2Sig.Init`, `2Sig.Sig`, `2Sig.Ver`), where $t = 1$ and $n = 2$. In this case, robustness is weakened due to the deployment of a 2-out-of-2 secret sharing. We refer the reader to [6, 25, 26, 38] for formal treatments of some two-party signature schemes.

HYBRID THRESHOLD SIGNATURE SCHEMES. A hybrid threshold signature scheme `HTSig` is the hybrid of a *two-party* signature scheme `2Sig` and a *multi-party* signature scheme `TSig` corresponding to the same underlying signature scheme `Sig`. In such a scheme, the privileges of the participants are weighted; for example, as shown in our first scheme `TPAKE-HTSig`, no signatures can be generated without the consent of a user, no matter how the servers collaborate. A hybrid threshold signature scheme `HTSig` consists of three algorithms (`HTSig.Init`, `HTSig.Sig`, `HTSig.Ver`). Taken as input a security parameter κ and the desired access structure in the underlying secret sharing scheme, the initialization algorithm `HTSig.Init` outputs a public key Y so that the corresponding private key X is shared among the participants (according to the given access structure). Taken as input a message m and the shares of the private key, `HTSig.Sig` outputs a signature σ. Typically, `HTSig.Ver` is the same as `Sig.Ver`. In Section

3.1 we will specify how to construct, and security definition of, `HTSig` for the subclass of signature schemes we are interested in.

THRESHOLD PASSWORD-AUTHENTICATED KEY EXCHANGE SCHEMES. A threshold password-authenticated key exchange scheme, or `TPAKE` for short, enables a set of servers to collaboratively authenticate a user using a password such that compromise of certain number of servers does not enable the adversary to conduct an off-line dictionary attack. A `TPAKE` scheme consists of two protocols: (`TPAKE.Init`, `TPAKE.Login`). Suppose κ is the main security parameter, t is the number of tolerated corrupt servers, and n is the number of servers. The basic functionality of `TPAKE.Init` is:

1. The servers collaboratively generate certain system-wide configurations (if any), some of which may be published.
2. Each user chooses her own password and sends its transformed version (e.g., after certain cryptographically-strong transformation) to the servers.

The invocation protocol `TPAKE.Login` is an algorithm of functionality that after a user successfully authenticates herself to the servers, the user shares a different on-the-fly session key with each of the servers. We will not present any concrete `TPAKE.Login` (indeed we don't pin down on any concrete `TPAKE`), instead we will present a specification (including a *security* and *robustness* definition) of `TPAKE`, namely `TPAKE.Login`, in Appendix A. Note that the first concrete `TPAKE` has been available [27].

3 The First Scheme: `TPAKE-HTSig`

Our first scheme is called `TPAKE-HTSig`, which stands for "'Threshold Password-Authenticated Key Exchange based' server-assisted Hybrid Threshold Signatures." `TPAKE-HTSig` is tailored for the following real-world scenario: a user having a networked device wants to enjoy a threshold protection of her private signing function by taking advantage of multiple servers, where the user device is powerful enough to efficiently compute modular exponentiations. The challenge in designing such a solution is that we must assure that only the user can activate threshold signing sessions even if the user device has been compromised (i.e., the secrets on it have been exposed). If we do not impose any restriction on the adversarial capability, an adversary having compromised a user device can always perfectly impersonate the user. Therefore, we assume that an adversary can compromise a user device only when the user software is inactive or the password is not in the device memory to which the adversary has access (such an assumption seems reasonable and popular [4, 23, 25, 27]).

3.1 How to Construct a `HTSig`

Given `Sig`=(`Sig.Init`, `Sig.Sig`, `Sig.Ver`) that belongs to the abovementioned subclass of signature schemes, we show how to construct a hybrid threshold signature scheme HTSig=(HTSig.Init, HTSig.Sig, HTSig.Ver) from its two-party variant 2Sig=(2Sig.Init, 2Sig.Sig, 2Sig.Ver) and multi-party variant TSig=(TSig.Init, TSig.Sig, TSig.Ver). Suppose (Y, X) is a user U's pair of public and private keys in Sig. Let $S_1, \cdots, S_n$ be the servers, and t be the number of tolerated corrupt servers.

The initialization algorithm HTSig.Init has the following two steps.

1. U uses a 2-out-of-2 secret sharing to share her private key X, namely $X \overset{(2,2)}{\longleftrightarrow} (X_U, X_S)$, where X_U is U's share. This step corresponds to 2Sig.Init.
2. U using an appropriate secret sharing scheme to share X_S, namely $X_S \overset{(t+1,n)}{\longleftrightarrow} (X_S^{(1)}, \cdots, X_S^{(n)})$, where $X_S^{(i)}$ $(1 \leq i \leq n)$ is S_i's share. This step corresponds to TSig.Init.

The signing algorithm HTSig.Sig has the following steps that correspond to the combination of 2Sig.Sig and TSig.Sig. Suppose m is the message that needs to be signed.

1. U generates a partial signature $\sigma_U = g_1'(m, X_U)$, where g_1' is an appropriate algorithm (which depends on the underlying signature scheme Sig).
2. S_{i_j} contributes its partial signature $\sigma_S^{(i_j)} = g_1(m, X_S^{(i_j)})$, where g_1 is an algorithm that depends on Sig, $1 \leq i_j \leq n$ for $1 \leq j \leq w$, and $t+1 \leq w \leq n$.
3. Given w valid partial signatures, anyone can compute $\sigma_S = g_2(\sigma_S^{(i_1)}, \cdots, \sigma_S^{(i_w)})$, where $1 \leq i_j \leq n$ for $1 \leq j \leq w$, $t+1 \leq w \leq n$, and g_2 is an appropriate algorithm that depends on Sig.
4. Given σ_U and σ_S, anyone can compute a signature $\sigma = g_2'(\sigma_U, \sigma_S)$, where g_2' is an appropriate algorithm that depends on Sig.

The verification algorithm HTSig.Ver is the same as Sig.Ver.

SECURITY OF HTSig. Security of a hybrid threshold signature scheme HTSig is captured by the following definition.

Definition 1. *(*Security of HTSig*; Informal Statement.) A hybrid threshold signature scheme* HTSig *is secure, if it is* unforgeable *and* robust. *A* HTSig *is* unforgeable *if there exists a polynomial-time simulator that is able to emulate* HTSig *while having access to a signing oracle corresponding to the underlying signature scheme* Sig. *More specifically, we require the existence of such a simulator in the following two cases.*

1. *X_U is exposed (i.e., the user device is compromised) but at most t servers are compromised (i.e., X_S is still secret).*
2. *X_S is exposed (i.e., at least $t+1$ servers are compromised) but X_U is secret.*

A HTSig *is* robust *if it remains to function in the presence of t corrupt servers.*

INSTANTIATIONS OF HTSig. We only investigate and sketch three HTSig schemes based on RSA, DSS, and Schnorr, respectively. An efficient HTSig can be based on RSA signature scheme [31, 5]. Suppose $N = PQ$ is a RSA modulus, $\langle e, N\rangle$ and $\langle d, N\rangle$ are a pair of RSA public and private keys so that $ed = 1 \bmod \phi(N)$.

A `HTSig` can be obtained as follows: the user first splits the secret exponent d into two pieces d_U and d_S (as in [25]), then the user shares d_S among the servers as in [30]. A signature is generated in a natural way.[3]

A `HTSig` can be based on DSS [11] by combining the schemes in [26, 16], although the resultant scheme is not very efficient. A much more efficient `HTSig` based on Schnorr [33] can be obtained by combing the schemes in [38, 18].

3.2 The Construction

We present a `TPAKE-HTSig` based on any `TPAKE` that satisfies the specification in Appendix A, and any `HTSig` that satisfies the specification in Section 3.1. The *glue* or *middleware* for integrating them together is a natural use of the on-the-fly session keys generated by the `TPAKE`, namely the keys are used as message authentication keys.

THE INITIALIZATION. This protocol consists of `TPAKE.Init` and `HTSig.Init`.

1. A user U and the servers $\{S_1, \cdots, S_n\}$ execute `TPAKE.Init`. (The basic functionality is that U sends certain transformed version of her password to the servers.)
2. A user U and the servers $\{S_1, \cdots, S_n\}$ execute `HTSig.Init`.
   (a) U executes $X \overset{(2,2)}{\longleftrightarrow} (X_U, X_S)$ and stores X_U on her device.
   (b) U executes $X_S \overset{(t+1,n)}{\longleftrightarrow} (X_S^{(1)}, \cdots, X_S^{(n)})$, and S_j $(1 \le j \le n)$ stores $X_S^{(j)}$.

THE INVOCATION. Let m be the message a user U wants to sign.

1. U initiates `TPAKE.Login` with a set of servers $(S_{i_1}, \cdots, S_{i_w})$. As a result, U holds w different on-the-fly session keys $sk_{i_1}, \cdots, sk_{i_w}$ that are also known to $S_{i_1}, \cdots, S_{i_w}$, respectively.
2. U and $(S_{i_1}, \cdots, S_{i_w})$ execute as follows. (This step is a combination of `HTSig.Sig` and the *middleware* – the application of the session keys.)
   (a) U generates a partial signature $\sigma_U = g_1'(m, X_U)$.
   (b) U sends $(m, \delta = f_{sk_{i_j}}(m))$ to server S_{i_j}, where $1 \le j \le w$ and f is a secure message authentication code.
   (c) S_{i_j}, where $1 \le j \le w$, verifies the integrity of the received (m, δ). If $\delta \ne f_{sk_{i_j}}(m)$, then it aborts; otherwise, it returns its partial signature $\sigma_S^{(i_j)} = g_1(m, X_S^{i_j})$ back to the user.
   (d) Given w valid partial signatures $\sigma_S^{(i_1)}, \cdots, \sigma_S^{(i_w)}$, U computes $\sigma_S = g_2(\sigma_S^{(i_1)}, \cdots, \sigma_S^{(i_w)})$.

[3] For real-world deployment, we have to ensure that there are no honest server that is unable to contribute its partial signatures in the signature generation process of Rabin [30]. This is so because that [30] recovers the secret share held by a dishonest server, which means that the share held by (for instance) a server that is under denial-of-service attack or temporarily down will be recovered, in spite of the fact that this server is never compromised. We are grateful to a reviewer for pointing out that Shoup's scheme [35] cannot be used in this setting.

(e) Given σ_U and σ_S, U computes the signature $\sigma = g'_2(\sigma_U, \sigma_S)$.

DISCUSSION 1. Any TPAKE satisfying the specification in Appendix A, and any HTSig satisfying the specification in Section 3.1, can be used to construct a secure TPAKE-HTSig (namely *plug-and-play*).

2. If HTSig is instantiated with RSA [30] (i.e., all the servers need to be involved in THE INVOCATION) and TPAKE is instantiated with the scheme of [27], then another layer of activation is needed (e.g., the $t+1$ servers that have authenticated a user activate the rest servers). We omit this issue in our security analysis, which nonetheless can be extended to take it into account.

3.3 Security Analysis

We focus on the unforgeability of TPAKE-HTSig; it is relatively easy to see that robustness of TPAKE-HTSig is ensured by the robustness of TPAKE and of HTSig.

Theorem. Assume the underlying HTSig and message authentication schemes are secure. If TPAKE-HTSig is broken, then TPAKE is broken. (A formal treatment is deferred to Appendix B.)

4 The Second Scheme: LW-TSig

Our second scheme LW-TSig, which stands for "Light-Weight server-assisted Threshold Signatures," is tailored for the following real-world scenario: A user, who has a smart card (or a hard-token in general) with no cryptographic co-processor, wants to enjoy threshold protection of her private signing function. This scheme is a simple, but useful, combination of a threshold signature scheme TSig and a message authentication mechanism.

4.1 The Construction

THE INITIALIZATION. This process involves a user U (who has a pair of public and private keys (Y, X)) and the servers $\{S_1, \cdots, S_n\}$, and is done in a secure environment (e.g., on a secure computer). After the process is finished, U stores $(key_1, \cdots, key_n)$ on her smart card, and S_j $(1 \leq j \leq n)$ holds (key_j, X_j).

1. U runs TSig.Init via executing $X \overset{(t+1,n)}{\longleftrightarrow} (X_1, \cdots, X_n)$, where X_j $(1 \leq j \leq n)$ is S_j's share.
2. U chooses and secretly sends a symmetric key key_j to S_j, where $1 \leq j \leq n$.

THE INVOCATION. This process involves a user U, a signature receiver, and the set of servers. Suppose U needs to sign a message m for the receiver.

1. U generates $\{\delta_j = f_{key_j}(m)\}_{j=1}^{n}$ and sends it to the signature receiver, where f is a secure message authentication code.
2. The signature receiver forwards (m, δ_j) to server S_j, where $1 \leq j \leq n$.

3. Server S_j ($1 \leq j \leq n$) checks whether the request (m, δ_j) is valid using the key key_j. If the request passes the test, S_j participates in the threshold signature generation protocol `TSig.Sig` (and perhaps sends its partial signature back to the signature receiver).
4. The signature receiver obtains a signature that can be verified using `TSig.Ver`.

4.2 Security Analysis

Theorem (Informal Statement). If both the message authentication scheme and the threshold signature scheme `TSig` are secure, then `LW-TSig` is secure.

5 A Taxonomy of Systems Protecting Signing Functions

5.1 The Terms

User storage-media. This specifies the type(s) of media a user used to store her secrets, and typically reflects the user's budget.

1. HUMAN-MEMORY. Human memory is used to remember passwords.
2. SOFT-TOKEN. A soft-token is a data structure that is stored on a networked device – typically a computer that can efficiently compute cryptographic operations like modular exponentiations. A soft-token may be further protected by a password.
3. HARD-TOKEN. A hard-token is a special-purpose hardware like smart card, to which an adversary may not have access. It may have a cryptographic co-processor, be tamper-resistant, and be protected with a password.
4. SOFT- & HARD-TOKEN. A user holds a soft-token (stored on a networked device) as well as a hard-token. One or both of them may be protected using a password.

Number of runtime key-shares. When the private signing function is active, the private key may exist in its entirety or be shared among multiple parties. A private key may exist in the form of shares before it is applied, as per [34]. Suppose at runtime a private key is of n-piece, then we consider three cases: $n = 1$ (i.e., ONE), $n = 2$ (i.e., TWO), and $n > 2$ (i.e., MULTIPLE).

5.2 The Taxonomy

We classify systems protecting private signing functions based on two factors: User storage-media – the x-axis, and Number of runtime key-shares – the y-axis. Let a coordinate (x, y) denote the corresponding system where $x \in$ {HUMAN-MEMORY, SOFT-TOKEN, HARD-TOKEN, SOFT- & HARD-TOKEN}, and $y \in$ {ONE, TWO, MULTIPLE}. Now we discuss the features of each of the 12 types of systems.

1. (HUMAN-MEMORY, ONE). In such a system, a user remembers a password which help her download her private key from a remote server [29] or remote servers [14, 22, 27]. Typically, in the transmission process the private key is protected using an on-the-fly session key generated using a secure password-authenticated key exchange protocol [7, 4, 23, 14, 27]. Note that such a system has a single point of failure.

2. (Human-memory, Two). There are further two types of systems. First, a user remembers a password from which a piece of her private key is derived, whereas the other piece of her private key is stored on a remote server. A typical scenario has been mentioned in [15, 6], and it has also been mentioned that such a system is subject to off-line dictionary attack. Second, a user remembers a password whereby he can activate two remote servers to collaboratively sign messages on her behalf. This is indeed a downsized (i.e., there are 2 remote servers) version of (Human-memory, Multiple) below.
3. (Human-memory, Multiple). There are also two types of systems. First, a user remembers a password from which a piece of her private key is derived, the other pieces of the private key are stored on remote servers. Typically, such a system is subject to off-line dictionary attack. Second, a user remembers a password whereby he can activate multiple remote servers to collaboratively sign messages on her behalf. Such a scheme is indeed a special case of our scheme `TPAKE-HTSig`.
4. (Soft-token, One). There are two types of systems. First, a user stores a password-protected private key on a device. Such a system is subject to off-line dictionary attack once the device is compromised. (Note that the variant introduced in [19] can prevent an off-line dictionary attack, but at the price of keeping the corresponding *public* key secret.) Second, a user deploys a mechanism called *forward security*, which guarantees that compromise of a private key at time t will not expose the private keys used at any past time $t' < t$. This notion was introduced in [1], and formalized in [3] which is followed by numerous papers (e.g., [24, 20]).
5. (Soft-token, Two). In such a system, a private key is split into two pieces such that one is stored on the user device as a soft-token, and the other is stored on a remote server. The authentication of a user to the remote server may be based on a password. The tailored constructions [9, 25, 8] fall into this category.
6. (Soft-token, Multiple). Our scheme `TPAKE-HTSig` is such a system, where a user splits her private key X into two pieces: X_U and X_S. The user stores her piece X_U on her networked device, and shares X_S among the remote servers.
7. (Hard-token, One). This is the traditional solution where a private key is stored on a (tamper-resistant) smart card with a cryptographic co-processor. Clearly, *forward security* can be incorporated.
8. (hard-token, two). There are further two systems. First, a user stores one share of her private key on her smart card, and the other on a remote server. The smart card or the share on it may be protected by a password. In such a system, the smart card is typically equipped with a cryptographic co-processor, which we want to avoid. Second, a user stores certain symmetric keys on her smart card, and shares her private key among two remote servers that collaborate in generating signatures. This is indeed a down-sized version of our scheme `LW-TSig`.
9. (Hard-token, Multiple). Our scheme `LW-TSig` is such a system, where a user's private key is shared among a set of servers and a smart card is used

to store some symmetric keys. The authentication of the user to the servers are based on symmetric key cryptography, so that the smart card does not need any cryptographic co-processor. Note that a password may be further deployed to protected a smart card.

10. (Soft- & Hard-token, One). The most recently introduced system [12, 21] fall into this category. The basic idea underlying *key insulation* [12] is that the system life time is divided into time periods, and a smart card is used only for updating private keys corresponding to a fixed public key. As a consequence, compromise of the user device, even at the runtime of the user software, will not compromise the private key corresponding to any past or future time period. An adversary compromising the smart card alone is still unable to generate any signatures that are valid with respect to the user's public key. The enhanced notion of *intrusion resilience* [21] assures *forward security* even if both the user device and smart card are compromised simultaneously. Note that a password may be used to protect the smart card.
11. (Soft- & Hard-token, Two). In such a system, a user device and smart card collaborate via a two-party signature scheme [6, 26, 38].
12. (Soft- & Hard-token, Multiple). Such a system could be seen as an extension of our schemes `TPAKE-HTSig` or `LW-TSig`, because the soft-token can hold a share of her private key (as per `TPAKE-HTSig`) and the smart card can hold $key_1, \cdots, key_n$ (as per `LW-TSig`). There are some interesting variants. For example, the smart card keeps the X_U in `TPAKE-HTSig` and the $key_1, \cdots, key_n$ in `LW-TSig`. Whenever the user needs to sign messages, she takes advantage of any (perhaps public) terminal to compute the partial signature with respect to X_U (which is of course given to the terminal and erased by the terminal after the session is finished), but the invocation of the remote servers are authenticated by the smart card (i.e., $key_1, \cdots, key_n$ never depart the smart card).

6 Conclusion

We presented two efficient and provably secure schemes for server-assisted threshold signatures, where the signing function is activated by a user (but using some enhanced authentication). The first one is tailored for the setting where the user device is able to efficiently compute modular exponentiations, and the second one is tailored for the setting where a user has a smart card without a cryptographic co-processor. We also presented a taxonomy of systems protecting private signing functions.

Acknowledgements. We are very grateful to the reviewers of RSA-CT'03 for their excellent comments that significantly improved this paper. We also thank Gene Tsudik for helpful comments.

References

1. R. Anderson. Invited Lecture. ACM CCS'97.
2. M. Bellare, J. Kilian, and P. Rogaway. The Security of Cipher Block Chaining. Crypto'94.
3. M. Bellare and S. Miner. A Forward-Secure Digital Signature Scheme. Crypto'99.
4. M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated Key Exchange Secure against Dictionary Attacks. Eurocrypt'2000.
5. M. Bellare and P. Rogaway. The exact security of digital signatures: How to sign with RSA and Rabin. Eurocrypt'96.
6. M. Bellare and R. Sandhu. The Security of Practical Two-Party RSA Signature Schemes. manuscript. 2001.
7. S. Bellovin and M. Merritt. Encrypted Key Exchange: Password-based Protocols Secure against Dictionary Attack. IEEE Security and Privacy'92.
8. D. Boneh, X. Ding, G. Tsudik, and C. Wong. A Method for Fast Revocation of Public Key Certificates and Security Capabilities. Usenix Security'01.
9. C. Boyd. Digital Multisignatures. Cryptography and Coding, pp 241-246, 1989.
10. R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. IEEE FOCS'01.
11. The US Digital Signature Standard, NIST, 1994.
12. Y. Dodis, J. Katz, S. Xu, and M. Yung. Strong Key-Insulated Signature Schemes. PKC'03, to appear.
13. P. Feldman. A Practical Scheme for Non-Interactive Verifiable Secret Sharing. IEEE FOCS'87.
14. W. Ford and B. Kaliski. Server-Assisted Generation of a Strong Secret from a Password. IEEE Workshops on Enabling Technologies'00.
15. R. Ganesan. Yaksha: Augmenting Kerberos with Public Key Cryptography. NDSS'95.
16. R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Robust Threshold DSS Signatures. Eurocrypt'96.
17. S. Goldwasser, S. Micali, and R. Rivest. A Digital Signature Scheme Secure against Adaptive Chosen-Message Attacks. SIAM J. Comput., (17)2, 1988, pp 281-308.
18. A. Herzberg, M. Jakobsson, S. Jarecki, H. Krawczyk, and M. Yung. Proactive Public Key and Signature Schemes. ACM CCS'97.
19. D. Hoover and B. Kausik. Software Smart Cards via Cryptographic Camouflage. IEEE Security and Privacy'99.
20. G. Itkis and L. Reyzin. Forward-Secure Signatures with Optimal Signing and Verifying. Crypto'01.
21. G. Itkis and L. Reyzin. SiBIR: Signer-Base Intrusion-Resilient Signatures. Crypto'02.
22. D. Jablon. Password Authentication using Multiple Servers. RSA-CT'01.
23. J. Katz, R. Ostrovsky, and M. Yung. Practical Password-Authenticated Key Exchange Provably Secure under Standard Assumptions. Eurocrypt'01.
24. H. Krawczyk. Simple Forward-Secure Signatures from any Signature Schemes. ACM CCS'00.
25. P. MacKenzie and M. Reiter. Networked Cryptographic Devices Resilient to Capture. IEEE Security and Privacy'01.
26. P. MacKenzie and M. Reiter. Two-Party Generation of DSA Signatures. Crypto'01.
27. P. MacKenzie, T. Shrimpton, and M. Jakobsson. Threshold Password-Authenticated Key Exchange. Crypto'02.

28. T. Pedersen. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. Crypto'91.
29. R. Perlman and C. Kaufman. Secure Password-based Protocol for Downloading a Private Key. NDSS'99.
30. T. Rabin. A Simplified Approach to Threshold and Proactive RSA. Crypto'98.
31. R. A. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. C. ACM. (21)2, 1978, pp 120-126.
32. R. Sandhu, M. Bellare, and Ravi Ganesan. Password-Enabled PKI: Virtual Smartcards versus Virtual Soft Tokens. PKI Research Workshop. 2002.
33. C. P. Schnorr. Efficient Signature Generation by Smart Cards. J. Cryptology, 1991.
34. A. Shamir. How to Share a Secret. C. ACM, 22(11):612–613, 1979.
35. V. Shoup. Practical Threshold Signatures. Eurocrypt'00.
36. J. Stern, D. Pointcheval, J. Malone-Lee, and N. Smart. Flaws in Applying Proof Methodologies to Signature Schemes. Crypto'02.
37. S. Xu and M. Yung. On the Dark Side of Threshold Cryptography. Financial Crypto'02.
38. S. Xu and M. Yung. A Provably Secure Two-Party Schnorr Signature Scheme. manuscript. 2002.

A A Specification of `TPAKE`

Instead of pinning down on any concrete construction, we give a specification of `TPAKE` so that any concrete construction satisfying this specification can be seamlessly embedded into our `TPAKE-HTSig`, namely *plug-and-play*. This specification is extended from the model of [27], which in turn builds on [4, 23].

PARTICIPANTS. There are two types of protocol participants: users and servers. Let $ID = Users \cup Servers$ be a non-empty set of protocol participants (i.e., principals), where $Servers = \{S_1, \cdots, S_n\}$. Each user $U \in Users$ has a secret password pwd_U, and each server $S \in Servers$ has a vector $pwd_S = [pwd_S(U)]_{U \in Users}$, where entry $pwd_S(U)$ is a password record. Let $Password$ be the dictionary from which the users' passwords are uniformly chosen (but the results can be easily extended to other password distributions). Let $D = |Password|$, which is typically small in the real world.

EXECUTION OF THE PROTOCOL. A protocol `TPAKE` (i.e., `TPAKE.Login`) is an algorithm that determines how principals behave in response to their environment. In the real world, each principal $P \in ID$ (modeled as a probabilistic polynomial-time algorithm) is able to execute `TPAKE` multiple times with different partners, and we model this by allowing unlimited number of instances of each principal. Instance i of principal P is denoted by Π_P^i.

To capture the security of `TPAKE`, we assume there is an adversary $\mathcal{A}$ that has complete control over the environment (mainly, the network), and thus provides the inputs to instances of principals. Since $\mathcal{A}$ controls the network and can deny service at any time, we do not consider any denial-of-service attack (as long as it has no security implication). We further assume the network (i.e., $\mathcal{A}$) performs aggregation and broadcast functions. In practice, on a point-to-point network, the protocol implementer would most likely have to implement these functionalities in some way, perhaps using a single intermediate (untrusted) node to aggregate and broadcast messages. Formally, the adversary is a probabilistic algorithm with a distinguished query tape. We assume that $\mathcal{A}$ may compromise up to t servers, and that the choice of these servers is static. In particular, we may assume that the choice is made before the initialization of TPAKE, and we may simply assume that the adversary has access to the secrets of the compromised servers. Queries written to the query tape are responded to by principals according to TPAKE. The adversary $\mathcal{A}$ can request the following queries.

1. Compromise(P). This results in that the secrets on $P \in ID$ are returned to $\mathcal{A}$. Note that (1) if $\mathcal{A}$ requests Compromise(U) then U's password is not returned to $\mathcal{A}$, and (2) at most t servers are compromised.
2. Send(P, i, M). This results in that message M is sent to instance Π_P^i, which then executes according to TPAKE (including the update of its internal state) and returns the result to $\mathcal{A}$. If this query causes Π_P^i to accept or terminate, this will also be shown to $\mathcal{A}$.
3. Execute($U, i, ((S_{j_1}, l_{j_1}), \cdots, (S_{j_w}, l_{j_w}))$), where $1 \leq j_z \leq n$ and $l_{j_z} \in \mathbb{N}$ for $1 \leq z \leq w$, $t+1 \leq w \leq n$. This results in the execution of TPAKE between Π_U^i ($U \in Users$) and ($\Pi_{S_{j_1}}^{l_{j_1}}, \cdots, \Pi_{S_{j_w}}^{l_{j_w}}$). The transcript is returned to $\mathcal{A}$.
4. Reveal(U, i, S_j). This results in that the session key held by Π_U^i corresponding to server S_j (i.e., $sk_{\Pi_U^i, S_j}$) is returned to $\mathcal{A}$.
5. Reveal(S_j, i). This results in that the session key held by $\Pi_{S_j}^i$ (i.e., $sk_{\Pi_{S_j}^i}$) is returned to $\mathcal{A}$.
6. Test(U, i, S_j). To response to this query, Π_U^i flips a coin b, and returns $sk_{\Pi_U^i, S_j}$ if $b = 0$, and a string that is drawn uniformly from the same space otherwise. A Test query (of either type) may be asked at any time during the execution of TPAKE, but may be asked only once.
7. Test(S_j, i). To response to this query, $\Pi_{S_j}^i$ flips a coin, and returns $sk_{\Pi_{S_j}^i}$ if $b = 0$, and a string that is drawn uniformly from the same space otherwise. A Test query (of either type) may be asked at any time during the execution of TPAKE, but may be asked only once.

Partnering using SIDs. Let sid be the concatenation of all messages sent and received by the user instance in its communication with the set of servers. (Note that this excludes the messages that are sent only between the servers, but not to the user.) A server instance that accepts holds a partner-id pid, session-id sid, and a session key sk. A user instance that accepts holds a session-id sid, a partner-id $pid = (pid_1, \cdots, pid_w)$ and the corresponding session keys $(sk_1, \cdots, sk_w)$. The instance Π_U^i ($U \in Users$) holding $(sid, (pid_1, \cdots, pid_w), (sk_1, \cdots, sk_w))$ and $\Pi_{S_j}^{l_j}$ ($S_j \in Servers$) holding (sid', pid', sk) are said to be partnered if there exists a unique $1 \leq z \leq w$ such that $pid_z = S_j$, $pid' = U$, $sid = sid'$, and $sk_z = sk$.

Freshness. A user instance/server pair (Π_U^i, S_j) is fresh if: (1) there has been no Compromise(S_j) query, (2) there has been no Reveal(U, i, S_j) query, and (3) if $\Pi_{S_j}^l$ is a partner of Π_U^i, there has been no Reveal(S_j, l) query. A server instance $\Pi^i_{S_j}$ is fresh if (1) there has been no COMPROMISE(S_j) query, (2) there has been no REVEAL(S_j, i) query, and (3) if Π^l_U is the partner to $\Pi^i_{S_j}$, there has been no REVEAL(U, l, S_j) query.

ADVANTAGE OF AN ADVERSARY $\mathcal{A}$ IN TPAKE. The goal of the adversary $\mathcal{A}$ is to guess the bit b. To formalize its advantage, let $\mathsf{Succ}^{\mathtt{TPAKE}}(\mathcal{A})$ be the event that $\mathcal{A}$ makes a single TEST query directed to some instance Π^i_P ($P \in ID$) that has terminated and is fresh, and eventually outputs a bit $b' = b$. The advantage of $\mathcal{A}$ attacking TPAKE is defined to be

$$\mathsf{Adv}^{\mathtt{TPAKE}}(\mathcal{A}) = 2 \cdot Pr[\mathsf{Succ}^{\mathtt{TPAKE}}(\mathcal{A})] - 1.$$

We say that $\mathcal{A}$ breaks TPAKE, if $\mathsf{Adv}^{\mathtt{TPAKE}}(\mathcal{A})$ is non-negligibly more than $\frac{q_{send}}{D}$, where q_{send} is the number of SEND(P, i, M) queries. *Security* and *robustness* requirements of a TPAKE is captured by the following definition.

Definition 2. *(*Security and robustness of TPAKE*) A* TPAKE *is said to be* secure *if for any probabilistic polynomial-time algorithm $\mathcal{A}$,* $\mathsf{Adv}^{\mathtt{TPAKE}}(\mathcal{A}) = \frac{q_{send}}{D} + \epsilon(\kappa^*)$*, where κ^* is a security parameter, q_{send} is the number of* SEND(P, i, M) *queries, D is the size of the dictionary from which the users choose their individual passwords, and $\epsilon(\cdot)$ is a negligible function. A* TPAKE *is said to be* robust *if compromise of t servers doesn't prevent it from functioning (i.e., outputting session keys).*

B Security Analysis of TPAKE-HTSig

B.1 Security Definition of TPAKE-HTSig

In order to capture the *security* (including *unforgeability* and *robustness*) of TPAKE-HTSig, we extend the above specification of TPAKE to capture the capability of an adversary $\mathcal{F}$ that intends to forge signatures in TPAKE-HTSig.

PARTICIPANTS. This is the same as in the specification of TPAKE, except for that each user $U \in Users$ has a pair of public and private keys (Y, X) corresponding to an appropriate signature scheme Sig.

EXECUTION OF THE PROTOCOL. A protocol TPAKE-HTSig is an algorithm that determines how principals behave in response to their environment. In the real world, each principal $P \in ID$ (modeled as a probabilistic polynomial-time algorithm) is able to execute TPAKE-HTSig multiple times with different partners, and we model this by allowing unlimited number of instances of each principal. Instance i of principal P is denoted by Π^i_P.

The adversary $\mathcal{F}$ in TPAKE-HTSig is similar to the adversary $\mathcal{A}$ in TPAKE, except for that $\mathcal{F}$ can only request the following queries.

1. COMPROMISE(P). This results in that the secrets on $P \in ID$ are returned to $\mathcal{F}$. Note that (1) if $\mathcal{F}$ requests COMPROMISE(U) then U's password is not returned to $\mathcal{F}$, and (2) at most t servers are compromised.

2. SEND(P, i, M). This results in that message M is sent to instance Π_P^i, which then executes according to TPAKE-HTSig (including the update of its internal state) and returns the result to $\mathcal{F}$.
3. EXECUTE$(U, i, ((S_{j_1}, l_{j_1}), \cdots, (S_{j_w}, l_{j_w}), m))$, where $1 \le j_z \le n$ and $l_{j_z} \in \mathbb{N}$ for $1 \le z \le w$, $t+1 \le w \le n$. This results in the execution of TPAKE-HTSig between Π_U^i ($U \in Users$) and ($\Pi_{S_{j_1}}^{l_{j_1}}, \cdots, \Pi_{S_{j_w}}^{l_{j_w}}$). The runtime transcript is returned to $\mathcal{F}$.
4. REVEAL(U, i, S_j). This results in that the session key held by Π_U^i corresponding to server S_j (i.e., $sk_{\Pi_U^i, S_j}$) is returned to $\mathcal{F}$.
5. REVEAL(S_j, i). This results in that the session key held by $\Pi_{S_j}^i$ (i.e., $sk_{\Pi_{S_j}^i}$) is returned to $\mathcal{F}$.

PARTNERING USING SIDS. This is the same as in the specification of TPAKE.

FRESHNESS. This is the same as in the specification of TPAKE.

THE SUCCESS PROBABILITY OF THE ADVERSARY $\mathcal{F}$ IN TPAKE-HTSig. Let $\mathsf{Succ}^{\mathtt{TPAKE\text{-}HTSig}}(\mathcal{F})$ be the event $\mathcal{F}$ outputs a valid (with respect to U's public key Y) signature σ on message m, while the following hold simultaneously:

1. there was no EXECUTE$(U, i, ((S_{j_1}, l_{j_1}), \cdots, (S_{j_w}, l_{j_w}), m)$ query;
2. if there is an un-compromised server S that ever outputs a partial signature on m, which means that an instance Π_S^l had received a message authentication tag $f_{sk}(m)$ that is valid with respect to the session key sk generated in a session of sid, then the following constraints apply:
   (a) (Π_U^i, S) is fresh, where Π_S^l and Π_U^i are partners in session sid;
   (b) There was no oracle query to either Π_S^l or Π_U^i for generating message authentication tag on message m. (Note that our security analysis remains sound even though we allow this type of oracle query that may not be available in real-world systems.)

Denote $Pr[\mathcal{F}\ \mathsf{Succ}]$ the probability that $\mathsf{Succ}^{\mathtt{TPAKE\text{-}HTSig}}(\mathcal{F})$ happens. We say that $\mathcal{F}$ breaks TPAKE-HTSig, if $Pr[\mathcal{F}\ \mathsf{Succ}]$ is non-negligibly more than $\frac{q_{send}}{D}$, where q_{send} is the number of SEND(P, i, M) queries.

We say that a TPAKE-HTSig is *robust* if compromise of t servers doesn't prevent it from functioning (i.e., outputting signatures).

B.2 The Theorems

Theorem 1. *Assume the underlying hybrid threshold signature scheme* HTSig *and message authentication scheme are secure. If there exists a probabilistic polynomial-time adversary $\mathcal{F}$ that breaks* TPAKE-HTSig *with probability $Pr[\mathcal{F}\ \mathsf{Succ}]$ that is non-negligibly more than $\frac{q_{send}}{D}$, then there exists a probabilistic polynomial-time algorithm $\mathcal{A}$ such that $\mathsf{Adv}^{\mathtt{TPAKE}}(\mathcal{A})$ is non-negligibly more than $\frac{q_{send}}{D}$.*

Proof. We assume that $\mathcal{F}$ is static and compromises servers $S_1, \cdots, S_t$ at the system initialization. We stress that either X_U or pwd_U, but not both, can be compromised.

There are two ways for $\mathcal{F}$ to forge signatures. First, it forges a message authentication tag $f_{sk}(m)$ with respect to some secret session key sk_j that is generated on-the-fly and held by an un-compromised server S_j, where $t+1 \leq j \leq n$. (This happens, for instance, when $\mathcal{F}$ succeeds in hitting U's password and therefore knows the on-the-fly message authentication key.) As a consequence, $\mathcal{F}$ that already knew X_U is able to get a signature by requesting the un-compromised server S_j to contribute a partial signature. Denote by "$\mathcal{F}$ $\mathsf{Succ\text{-}AU}$" the event that $\mathcal{F}$ succeeds in forging a message authentication tag that is accepted by an un-compromised server S_j, and by $Pr[\mathcal{F}\ \mathsf{Succ\text{-}AU}]$ the probability that this event happens. Second, it forges a signature without forging any message authentication tag. This implies that $\mathcal{F}$ is able to output a signature by having access to: (1) $X_S^{(1)}, \cdots, X_S^{(t)}$, and perhaps X_U, or (2) X_S which is obtained by compromising a quorum number of servers.

Note that

$$\begin{aligned} Pr[\mathcal{F}\ \mathsf{Succ}] &= Pr[\mathcal{F}\ \mathsf{Succ}|\mathcal{F}\ \mathsf{Succ\text{-}AU}] \cdot Pr[\mathcal{F}\ \mathsf{Succ\text{-}AU}] + \\ &\qquad Pr[\mathcal{F}\ \mathsf{Succ}|\neg\mathcal{F}\ \mathsf{Succ\text{-}AU}] \cdot Pr[\neg\mathcal{F}\ \mathsf{Succ\text{-}AU}] \\ &= Pr[\mathcal{F}\ \mathsf{Succ\text{-}AU}] + Pr[\mathcal{F}\ \mathsf{Succ}|\neg\mathcal{F}\ \mathsf{Succ\text{-}AU}] \cdot Pr[\neg\mathcal{F}\ \mathsf{Succ\text{-}AU}]. \end{aligned}$$

In order to prove the theorem, we prove two lemmas. By Lemma 1 we show $Pr[\mathcal{F}\ \mathsf{Succ}|\neg\mathcal{F}\ \mathsf{Succ\text{-}AU}]$ is negligible, which means the probability that $\mathcal{F}$ succeeds in forging a signature without forging a message authentication tag is negligible. According to the assumption that $Pr[\mathcal{F}\ \mathsf{Succ}]$ is non-negligibly more than $\frac{q_{send}}{D}$, we know that $Pr[\mathcal{F}\ \mathsf{Succ\text{-}AU}]$ is non-negligibly more than $\frac{q_{send}}{D}$. By Lemma 2 we show that if $Pr[\mathcal{F}\ \mathsf{Succ\text{-}AU}]$ is non-negligibly more than $\frac{q_{send}}{D}$, then there is a probabilistic polynomial-time algorithm $\mathcal{A}$ that is able to break `TPAKE` with probability non-negligibly more than $\frac{q_{send}}{D}$. Therefore, the theorem holds.

Lemma 1. *If* `HTSig` *is secure, then* $Pr[\mathcal{F}\ \mathsf{Succ}|\neg\mathcal{F}\ \mathsf{Succ\text{-}AU}]$ *is negligible.*

Proof. Suppose $Pr[\mathcal{F}\ \mathsf{Succ}|\neg\mathcal{F}\ \mathsf{Succ\text{-}AU}]$ is non-negligible, then we show that there is a probabilistic polynomial-time algorithm $\mathcal{F}'$ that is able to break `HTSig` with non-negligible probability. According to Definition 1, we know that there is a simulator that is able to emulate `HTSig` by having access to a centralized signing oracle in the underlying signature scheme `Sig`. The basic idea underlying the proof of this lemma is that $\mathcal{F}'$ maintains a `TPAKE-HTSig` environment by establishing a `TPAKE` sub-environment that is beyond the scope of `HTSig`. Specifically, $\mathcal{F}'$ executes as follows.

1. $\mathcal{F}'$ emulates THE INITIALIZATION of `TPAKE-HTSig` as follows.
   (a) $\mathcal{F}'$ executes the first step of THE INITIALIZATION in `TPAKE-HTSig`; namely, `TPAKE.Init`. As a consequence, $\mathcal{F}'$ knows all the configurations of the servers' and the passwords of the users'.
   (b) $\mathcal{F}'$ emulates the second step of THE INITIALIZATION in `TPAKE-HTSig`; namely, `HTSig.Init`. The simulation in the case that X_U is compromised typically differs from the simulation in the case that X_S is compromised, but the existence of such simulators is guaranteed.

2. $\mathcal{F}'$ emulates THE INVOCATION of `TAPKE-HTSig` as follows.
   (a) Suppose X_U has been compromised.
      i. $\mathcal{F}'$ executes `TPAKE.Login` on behalf of the servers. If an authentication session is successfully finished, then $\mathcal{F}'$ holds the corresponding message authentication keys.
      ii. The user sends a signing request to $\mathcal{F}'$.
      iii. If the signing request is valid, $\mathcal{F}'$ returns the simulated partial signatures corresponding to the servers chosen by the user. This is done as if $\mathcal{F}'$ is simulating a multi-party signature scheme `TSig` by having oracle access to a signing oracle.
      iv. The user computes the signature after obtaining w valid partial signatures from $\mathcal{F}'$, where $t+1 \leq w \leq n$.
   (b) Suppose X_S has been compromised. In this case, $\mathcal{F}'$ is indeed the simulator that emulates the two-party signature scheme `2Sig` corresponding to the underlying signature scheme `Sig`.

Since $\mathcal{F}$ does not forge any message authentication tag, all of the signatures that $\mathcal{F}'$ has obtained are also available to the adversary $\mathcal{F}$. As a consequence, any forgery by $\mathcal{F}$ in the simulated `TPAKE-HTSig` is a forgery by $\mathcal{F}'$ in `HTSig`.

Lemma 2. *Suppose the message authentication code is secure. If $Pr[\mathcal{F}$ Succ-AU$]$ is non-negligibly more than $\frac{q_{send}}{D}$, then there exists a probabilistic polynomial-time algorithm $\mathcal{A}$ that breaks* `TPAKE` *with probability non-negligible more than* $\frac{q_{send}}{D}$.

Proof. The basic idea underlying the proof of this lemma is that $\mathcal{A}$ maintains a `TPAKE-HTSig` environment while having access to a `TPAKE` sub-environment. In particular, $\mathcal{A}$ maintains a `HTSig` sub-environment that is beyond the `TPAKE`.

For notational reason, let SID be the set of the sid's that correspond to those sessions whereby a non-null session key is defined in each of them. We can then define a total order on the elements of SID. Denote by $N = |SID|$, then we assume that each element in SID can be identified by $i \in \{1, \cdots, N\}$.

Now we present the algorithm $\mathcal{A}$.

1. $\mathcal{A}$ chooses a pair of public and private keys (Y, X) and executes the second step of THE INITIALIZATION in `TPAKE-HTSig` (to share X among the user and the servers).
2. $\mathcal{A}$ chooses a random $z \in \{1, \cdots, N\}$ corresponding to a session between $\Pi^i_{U'}$ and $\{S_{j_1}, \cdots, S_{j_w}\}$, where $1 \leq j_z \leq n$ for $1 \leq z \leq w$, $t+1 \leq w \leq n$. $\mathcal{A}$ uniformly chooses $j \in \{j_1, \cdots, j_w\} \setminus \{1, \cdots, t\}$ and executes $\textsc{Test}(U, i, S_j)$ in the `TPAKE` sub-environment. The environment flips a coin b, and returns $sk = sk_{\Pi^i_U, S_j}$ if $b = 0$, and a random string otherwise. For each of the other session keys, $\mathcal{A}$ requests a REVEAL to get it. Moreover, $\mathcal{A}$ answers $\mathcal{F}$'s queries in the emulated `TPAKE-HTSig` as follows.
   (a) For a query $\textsc{Compromise}(P)$, $\mathcal{A}$ returns the corresponding secrets to $\mathcal{F}$.
   (b) For a query $\textsc{Send}(P, i, M')$, there are two cases.

i. If the query is for TPAKE, $\mathcal{A}$ forwards $\text{Send}(P, i, M)$ to the TPAKE sub-environment and returns its response to $\mathcal{F}$, where M is well-defined due to the modular composition of TPAKE-HTSig.

ii. If the query is for HTSig, $\mathcal{A}$ executes according to the specification of TPAKE-HTSig and returns the result back to $\mathcal{F}$.

(c) For a query $\text{Execute}(U, i, ((S_{j_1}, l_{j_1}), \cdots, (S_{j_w}, l_{j_w}), m)$, where $1 \leq j_z \leq n$ and $l_{j_z} \in \mathbb{N}$ for $1 \leq z \leq w$, $\mathcal{A}$ forwards $\text{Execute}(U, i, ((S_{j_1}, l_{j_1}), \cdots, (S_{j_w}, l_{j_w})))$ to the TPAKE sub-environment, and emulates the runtime beyond TPAKE. $\mathcal{A}$ returns the response from the TPAKE sub-environment and the emulated transcript that is beyond the TPAKE back to $\mathcal{F}$.

(d) For a query $\text{Reveal}(U, i, S_j)$, $\mathcal{A}$ can answer it with probability at least $1 - \frac{1}{(n-t)N}$ since it has obtained the corresponding session key.

(e) For a query $\text{Reveal}(S_j, i)$, $\mathcal{A}$ can answer it with probability at least $1 - \frac{1}{(n-t)N}$ since it has obtained the corresponding session key.

Now, if $\mathcal{F}$ succeeds in forging a message authentication tag that is valid with respect to the key sk returned by the $\text{Test}(\cdot, \cdot, \cdot)$ oracle in the TPAKE sub-environment, then $\mathcal{A}$ returns 0 (meaning that it bets $sk = sk_{\Pi_U^i, S_j}$ or $b = 0$); otherwise, $\mathcal{A}$ returns a random bit.

We claim that if $b = 1$, which means that the sk obtained from the $\text{Test}(\cdot, \cdot, \cdot)$ oracle is a random string, then $\mathcal{F}$ has negligible probability in forging a valid message authentication tag with respect to $f_{sk}(\cdot)$. Otherwise, the message authentication scheme is broken. Therefore, $Pr[\mathcal{F}\ \mathsf{Succ\text{-}AU}|b = 1]$ is negligible.

Now we compute the probability that $\mathcal{A}$ successfully hits the bit b when $\mathcal{F}$ succeeds in forging a message authentication tag; namely,

$$\begin{aligned} Pr[\mathcal{A}\ \mathsf{Succ}|\mathcal{F}\ \mathsf{Succ\text{-}AU}] &= Pr[b = 0|\mathcal{F}\ \mathsf{Succ\text{-}AU}] \\ &= \frac{Pr[b = 0] \cdot Pr[\mathcal{F}\ \mathsf{Succ\text{-}AU}|b = 0]}{Pr[\mathcal{F}\ \mathsf{Succ\text{-}AU}]}. \end{aligned}$$

Therefore, we have

$$\begin{aligned} Pr[\mathcal{A}\ \mathsf{Succ}] &= Pr[\mathcal{A}\ \mathsf{Succ}|\mathcal{F}\ \mathsf{Succ\text{-}AU}] \cdot Pr[\mathcal{F}\ \mathsf{Succ\text{-}AU}] + \\ &\qquad Pr[\mathcal{A}\ \mathsf{Succ}|\neg\mathcal{F}\ \mathsf{Succ\text{-}AU}] \cdot Pr[\neg\mathcal{F}\ \mathsf{Succ\text{-}AU}] \\ &= Pr[b = 0] \cdot Pr[\mathcal{F}\ \mathsf{Succ\text{-}AU}|b = 0] + 0.5(1 - Pr[\mathcal{F}\ \mathsf{Succ\text{-}AU}]) \\ &= Pr[\mathcal{F}\ \mathsf{Succ\text{-}AU}] - Pr[b = 1] \cdot Pr[\mathcal{F}\ \mathsf{Succ\text{-}AU}|b = 1] + \\ &\qquad 0.5(1 - Pr[\mathcal{F}\ \mathsf{Succ\text{-}AU}]) \\ &= 0.5(1 + Pr[\mathcal{F}\ \mathsf{Succ\text{-}AU}] - Pr[\mathcal{F}\ \mathsf{Succ\text{-}AU}|b = 1]) \end{aligned}$$

and

$$\begin{aligned} \mathsf{Adv}^{\mathtt{TPAKE}}(\mathcal{A}) &= 2 \cdot Pr[\mathcal{A}\ \mathsf{Succ}] - 1 \\ &= Pr[\mathcal{F}\ \mathsf{Succ\text{-}AU}] - Pr[\mathcal{F}\ \mathsf{Succ\text{-}AU}|b = 1], \end{aligned}$$

which is non-negligibly more than $\frac{q_{send}}{D}$.

As a final comment, we note that the tight reduction in security comes from the fact that the simulator is of expected polynomial-time (with an expected slowdown factor no greater than $(n-t)N$).