A Scalable and Secure Cryptographic Service

Shouhuai Xu[1] and Ravi Sandhu[2]

[1] Department of Computer Science, University of Texas at San Antonio
shxu@cs.utsa.edu
[2] Department of Information and Software Engineering, George Mason University
sandhu@gmu.edu

Abstract. In this paper we present the design of a scalable and secure cryptographic service that can be adopted to support large-scale networked systems, which may require strong authentication from a large population of users. Since the users may not be able to adequately protect their cryptographic credentials, our service leverages some better protected servers to help fulfill such authentication needs. Compared with previous proposals, our service has the following features: (1) it incorporates a 3-factor authentication mechanism, which facilitates compromise detection; (2) it supports immediate revocation of a cryptographic functionality in question; (3) the damage due to the compromise of a server is contained; (4) it is scalable and highly available.

Keywords: cryptographic service, scalability, security, compromise detection, compromise confinement, availability.

1 Introduction

Large-scale networked systems, such as peer-to-peer and grid systems, must be adequately protected; otherwise they may be abused or exploited to do more harm than good — Distributed Denial-of-Service (DDoS) attacks are just an example. An important aspect of secure large-scale networked systems is to enforce strong authentication, which would require a large population of users to utilize some cryptosystems such as digital signatures. Due to the very nature of cryptography, assurance offered by such authentications perhaps cannot be any better than security (or secrecy) of the corresponding cryptographic keys or functionalities. This is because compromise of a cryptographic key would allow the adversary to perfectly impersonate the victim user. The threat is amplified by the fact that average users often do not have the expertise or skill to secure their own computers, which may be justified by the fact that there have been many Botnets that consist of many compromised computers.

1.1 Our Contributions

We present the design of a scalable and secure cryptographic service that can be adopted to support large-scale networked systems, which require strong authentication from a large population of users. Our approach leverages some better protected servers to help accommodate such authentication needs. Specifically, the servers may provide cryptographic services to the users, perhaps for some economic or social incentives. Compared with previous proposals for a similar purpose, our service has the following features:

* It incorporates a 3-factor authentication mechanism so that a signature may be produced in a certain way when a request: (1) presents a valid password (i.e., what you know); (2) is initiated from a party having access to the user's soft-token (i.e., what you have); (3) presents a valid fresh one-time secret (i.e., whether you *always* have access to the soft-token). As a result, the resulting service provides a *compromise detection* capability that may be of independent value.
* It supports a convenient *key disabling* that can be done using a standard username/password authentication. This is useful, for instance, when a user's device is stolen on a business trip, the user can disable its private key without having access to a backup of the content on its stolen device.

* The damage due to the compromise of a server is confined to a subset of the users subscribing to its service. Furthermore, the users associated with a compromised server do not have to re-initialize their private keys, unless they suspect that their own machines might have been compromised.
* It is scalable due to its "decentralized" nature (i.e., each server only serves a subset of users). It is highly available since a single server is enough to help a user fulfill its task. While we do not explore the details of utilizing the state machine approach [30] to securely replicate a server, it should not be difficult to extend our solution to fulfill such replications. In particular, in a related prior scheme we proposed [31], threshold cryptosystems have been adopted to fulfill distributed password-based authentication and signing.

1.2 Related Work

The simplest approach to securing cryptographic keys or functionalities is to let each user utilize some tamper-resistant hardware device. The industry has started to provide machines equipped with Trusted Platform Module (TPM) as specified by the Trusted Computing Group (`www.trustedcomputinggroup.com`). However, there are many legacy computers that need be better protected, meaning that alternate solutions are still useful.

One alternate approach to protecting cryptographic keys is to encrypt a key with a password; this is indeed widely deployed in real-life systems. However, the resulting security assurance is quite weak because, once a computer is compromised, the adversary can obtain the cryptographic keys without even conducting an off-line dictionary attack. Another approach is to let a user store its cryptographic key in a remote server, and download the key from the server after a password-based authentication [26]. This approach still allows the adversary, who can compromise a user's computer, to obtain the private key in question. Moveover, the remote server has to be trusted.

Our scheme follows the paradigm of "cryptography as a service" [13]. (This paradigm is different from the server-aided protocols [25, 2], which were motivated to utilize a computationally powerful server to help a computationally poor device conduct expensive cryptographic computations.) In particular, we adopt as a starting point the proposal due to MacKenzie and Reiter [23], which follows [10, 17]. The basic idea common to [23, 10, 17] as well as a similar result [9] is to split a private key into two shares such that one share (often called "soft-token") is stored at the user device, and the other is stored at a remote server. The shares may be protected with some tricky methods. Unlike the approaches mentioned above, the resulting scheme does not have any single point of failure.

For concreteness, we assume in our presentation that a service provider deploys a single server. In practice, the server may deploy some advanced cryptographic constructs such as forward-security cryptosystems [1, 3, 20], key-insulated cryptosystems [15] and intrusion-resilient cryptosystems [21]. Moreover, password authentications can also be extended to accommodate the case of multiple servers [24, 31]. Such extensions are orthogonal to the approach of the present paper, meaning that they can be seamlessly integrated together in practice for a better protection.

Outline: In Section 2 we present some necessary cryptographic preliminaries. In Section 3 we introduce the soft-token system model. In Section 4 we present a building block, which is utilized in Section 5 to construct the full-fledged service scheme. We conclude the paper in Section 6.

2 Cryptographic Preliminaries

Let k be the main security parameter (e.g., $k = 160$), and λ be a secondary security parameter for public keys (e.g., $\lambda = 1024$ means we use 1024-bit RSA moduli). Let H (with an additional subscript as needed) be a hash function that, unless otherwise stated, is assumed to behave like a random oracle [16, 5] with range $\{0,1\}^k$. A function $\epsilon : \mathbb{N} \rightarrow \mathbb{R}^+$ is negligible if for any c there exists κ_c such that $\forall \kappa > \kappa_c$ we have $\epsilon(\kappa) < 1/\kappa^c$.

Pseudorandom functions. A pseudorandom function (PRF) family $\{f_v\}$ parameterized by a secret value v has the following property [18]: It is computationally infeasible to distinguish an oracle for f_v, where v is uniformly chosen at random, from an oracle for a random function (with the same domain and range).

Message authentication codes. We assume the standard property of message authentication codes (MACs): If the key a is unknown, then given multiple pairs $\langle m_i, MAC_a(m_i)\rangle$ where the m_i's may be adaptively chosen, it is computationally infeasible to compute any pair $\langle m, MAC_a(m)\rangle$ where $m \neq m_i$.

Public key cryptosystems. An public key cryptosystem $\mathcal{E}$ is a triple (GEN, E, D) of polynomial-time algorithms, where the first two are probabilistic. GEN, taking as input 1^λ, outputs a key pair (pk, sk). E, taking as input a public key pk and a message m, outputs an encryption c for m. D, taking as input a ciphertext c and a private key sk, returns a message m when c is valid and $\perp$ otherwise. We assume an encryption scheme that is secure against adaptive chosen-ciphertext attack [28]. Basically, an attacker $\mathcal{A}$ is given pk and allowed to query the decryption oracle. At some point $\mathcal{A}$ generates two equal length strings X_0 and X_1 and sends them to a test oracle, which chooses $b \in_R \{0, 1\}$ and returns $Y = E_{pk}(X_b)$. Then $\mathcal{A}$ continues querying the decryption oracle, with the restriction that it cannot query the decryption of Y. Finally, $\mathcal{A}$ output b'. We say $\mathcal{A}$ succeeds if $b = b'$. Practical schemes are available [6, 12].

Digital signature schemes. A digital signature scheme $\mathcal{S}$ is a triple (GEN, S, V) of polynomial-time algorithms, where the first two are probabilistic. GEN, taking as input 1^λ, outputs a key pair (pk, sk). S, taking as input a message m and a private key sk, outputs a signature σ for m. V, taking as input a message m, a public key pk, and a candidate signature σ, returns $b = 1$ if σ is a valid signature for m and $b = 0$ otherwise. We assume a signature scheme that is existentially unforgeable under adaptive chose-message attack [19]: a forger is given pk; it is allowed to query a signature oracle on messages of its choice; it succeeds if it outputs a valid signature for m that is not one of the messages signed before.

3 Model and Goals

System model. There are a set of users and a set of servers that are operated by one or more service providers. As in a standard Public Key Infrastructure (PKI), a server has a pair of public and private keys, and so is a user. All the users and servers are modeled as probabilistic polynomial-time algorithms. A user splits its private key into two shares after a certain cryptographic transformation; roughly speaking, one share is stored on the user side (perhaps being encrypted with a password) and the other is stored on a remote server. A soft-token is a data structure a user stores. A soft-token (containing a user-side key share) may be stateful, so we may denote by $token^{(i)}$ the soft-token after the i^{th} transaction in which it is utilized.

A server has two interfaces: one for producing signatures and the other for disabling private keys. The resulting signatures can be verified using the public keys of the users, and thus can be used for authentication in higher-layer applications such as large-scale networked systems mentioned above. In order for a user to produce a signature corresponding to its private key, the user conducts an interaction with a server, which collaborates with the claimed user only when the user successfully authenticates itself. In order for a user to disable its private key, the user needs to succeed in a certain authentication operation. A server maintains a database for recording relevant information that would allow the service provider to take actions (e.g., gathering payment when the service is payment-based).

Adversary. We consider an adversary who may have control over the network to some extent. The adversary may compromise certain resources including a user's soft-token, a user's password, and a server's private key. The adversary may break into a user's device when the client software is *active*, which means that we consider an adversary that is strictly more powerful than the adversary considered in previous soft-token systems. We assume that integrity of the server side (e.g., the database) is guaranteed, even if an adversary may be able to compromise the server's private key. This also models the situation where a semi-trusted server may be honest in performing the protocol, but curious about the users' private keys.

Goals. Let κ be the primary security parameter. A cryptographic service should have the following properties:

* Abuse prevention. Consider a fixed pair of $\langle user, server \rangle$, where the $user$ possesses a soft-token $token$ and a password pwd. Denote by $\mathcal{ADV}(R)$ the type of adversary who succeeds in capturing the resource elements of $R \subseteq \{token, server, pwd\}$. When we say that an adversary $\mathcal{ADV}$ has access to $token$ we mean that $token$ is always available to $\mathcal{ADV}$; when we say that $\mathcal{ADV}$ has access to $\neg token^{(i)}$ we mean that $\mathcal{ADV}$ does not have access to $token^{(i)}$ but perhaps has access to $token^{(j)}$ for $0 \leq j < i$. Specifically,
  1. An adversary of type $\mathcal{ADV}\{server, pwd\}$ can forge signatures that are valid with respect to the user's public key with a negligible probability in κ.
  2. An adversary of type $\mathcal{ADV}\{token, server\}$ can forge signatures that are valid with respect to the user's public key only when the adversary succeeds in off-line dictionary guessing the user's password.
  3. An adversary of type $\mathcal{ADV}\{token\}$ can forge signatures that are valid with respect to the user's public key with probability negligibly more than $q/|\mathbb{D}|$ after q invocations of the server, where $\mathbb{D}$ is the dictionary from which the user's password is randomly drawn.
  4. An adversary of type $\mathcal{ADV}\{token, pwd\}$ can output — with only a probability negligible in κ — signatures that are valid with respect to the user's public key after the user's private key is disabled.
  5. An adversary of type $\mathcal{ADV}\{\neg token^{(i)}, pwd\}$ can output — with only a probability negligible in κ — signatures that are valid with respect to the user's public key after the user finishes the i^{th} transaction and before the user initiates the $(i+1)^{th}$ transaction.
* Compromise detection. The system itself can detect the compromise that an adversary has succeeded in impersonating the user for producing signatures.
* Immediate revocation. A user can request a server to disable its private key by executing a standard username/password authentication.
* Compromise confinement. The impact due to the compromise of a server is contained to a *subset* of the users subscribing to its service. Moreover, these users do not have to re-initialize their private keys.
* Scalability. The system can serve a large population of users.
* High availability. The system is highly available, even if some servers are under DDoS attacks.

4 Building Block: A Single Server Soft-Token Scheme

In this section we present a building block, which is extended from a scheme presented in [23] and will be incorporated into our full-fledged scheme in Section 5. Suppose the server's public data (e.g., public key) are available to the users, and consider a fixed pair of $\langle user, server \rangle$. The user runs the initialization process to generate a soft-token $token^{(0)}$ using its public and secret data as well as the server's public data. In the i^{th} transaction ($i = 1, 2, \cdots$), the user, who has access to $token^{(i-1)}$ that is generated in the $(i-1)^{th}$ transaction or in the initialization process when $i = 1$, signs a message by interacting with the server. The server collaborates with the claimed user only when the user presents the password and the state information ϑ chosen by the server in the last transaction. At the end of the i^{th} transaction, the user obtains the signature, generates a new token $token^{(i)}$ using the cryptographic state information ϑ chosen by the server, and erases $token^{(i-1)}$. Correspondingly, the server tracks the changes of the ϑ's in its database.

Denote a user's public key by $pk_{user} = \langle e, N \rangle$ and private key by $sk_{user} = \langle d, N, \phi(N) \rangle$, where $ed = 1 \bmod \phi(N)$, N is the product of two large prime numbers, and ϕ is the Euler totient function.

In the standard encode-then-sign paradigm, the signature sig on message m is $S_{\langle d,N,\phi(N)\rangle}(m) = \langle r, s\rangle$, where $r \in_R \{0,1\}^{len_{pad}}$, $s = (\mathsf{encode}(m,r))^d \bmod N$ for some encoding function encode. A signature $\langle r, s\rangle$ can be verified by checking if $s^e = \mathsf{encode}(m,r) \bmod N$. The function encode could be either deterministic (e.g., $len_{pad} = 0$ in the case of hash-and-sign [14]) or probabilistic (e.g., PSS [7]), these types of signatures were proven secure against adaptive chosen-message attacks in the random oracle model. The basic idea for splitting the private key d is to let $d_1 + d_2 = d \bmod \phi(N)$.

The scheme has the following components.

Token initialization. Suppose $H_1 : \{0,1\}^* \longrightarrow \{0,1\}^k$ and $f : \{0,1\}^* \rightarrow \{0,1\}^{\lambda+k}$. The inputs are the server's public encryption key pk_{server}, the user's password pwd, the user's public key $pk_{user} = \langle e, N\rangle$ and private key $sk_{user} = \langle d, N, \phi(N)\rangle$. The initialization proceeds as follows.

$$\begin{array}{lll}
uid & = & username \\
v & \in_R & \{0,1\}^k \\
a & \in_R & \{0,1\}^k \\
b & = & H_1(pwd) \\
d_1 & = & f(v, pwd) \\
d_2 & = & d - d_1 \bmod \phi(N) \\
\tau & = & E_{pk_{server}}(\langle a, b, uid, d_2, N\rangle) \\
ct & = & 0 \\
st & \in_R & \{0,1\}^k \\
token & = & (ct, st, v, a, \tau, e, N, pk_{server})
\end{array}$$

The user chooses its own uid – a memorizable string such as its email address. The soft-token is $token = (ct, st, v, a, \tau, e, N, pk_{server})$, where ct is an incremental counter indicating the serial number of a transaction (which is used for simplifying the description), st is the state information that will be chosen by the server (for the time being of $ct = 0$, it is just a placeholder). All the other values, including b, d, d_1, d_2, $\phi(N)$, and pwd, are erased.

Server database. The server maintains a database of $\Upsilon = (\tau, uid, count, \vartheta, m, r)$, which represents a transaction corresponding to $\tau = E_{pk_{server}}(\langle a, b, uid, d_2, N\rangle)$, where uid is obtained after receiving the first service request, $count$ is an incremental counter (with initialized value zero) maintained by the server. There are two types of operations regarding the database. First, $\mathsf{append}(\tau, uid, count, \vartheta, m, r)$ appends the tuple $(\tau, uid, count, \vartheta, m, r)$ into the database. Second, $\mathsf{last}(\tau, uid, count, \vartheta, m, r)$ returns either the tuple $(\tau, uid, count, \vartheta, m, r)$ corresponding to the *last* transaction corresponding to τ, or NULL meaning that the token corresponding to τ has never been used before.

Signing protocol. The user runs the client software, which prompts the user to enter the password pwd, to get the to-be-signed message m as well as the soft-token $token = (ct, st, v, a, \tau, e, N, pk_{server})$. The protocol is depicted in Fig. 1. Let us briefly explain the functions of the protocol components:

* β is a value showing that the user knows the password pwd;
* ρ_1 and ρ_2 are two one-time pads chosen by the user, and will be used by the server to encrypt the state information ϑ and the partial signature σ (produced using the partial private key d_2), respectively;
* ρ_3 is a one-time message authentication key that allows the user to detect compromise of $token$ because, otherwise, the adversary could tamper with θ while keeping η intact;
* r is a len_{pad}-bit random string used in the encoding function;
* γ is the encryption of m, r, β, st, ρ_1, ρ_2, and ρ_3;
* δ and φ are message authentication codes computed using a and ρ_3, respectively. (Note that both δ and φ are not for preventing abuses, but for detecting attacks.)

Key disabling protocol. In order to disable its private key, the user authenticates itself to the server by conducting a standard username/password authentication protocol corresponding to uid/pwd. The

USER		SERVER
$token = (ct, st, v, a, \tau, e, N, pk_{server}),\ \beta = H_1(pwd)$		
$\rho_1 \in_R \{0,1\}^k,\ \rho_2 \in_R \{0,1\}^\lambda,\ \rho_3 \in_R \{0,1\}^k,\ r \in_R \{0,1\}^{len_{pad}}$		
$\gamma = E_{pk_{server}}(\langle m, r, \beta, st, \rho_1, \rho_2, \rho_3 \rangle)$		
$\delta = MAC_a(\langle \gamma, \tau \rangle)$		
	(γ, τ, δ) $\longrightarrow$	
		abort IF τ has been disabled
		$\langle a, b, uid, d_2, N \rangle = D_{sk_{server}}(\tau)$
		abort IF $MAC_a(\langle \gamma, \tau \rangle) \neq \delta$
		$\Upsilon = \mathsf{last}(\tau, uid, count, \vartheta', m', r')$
		$\langle m, r, \beta, st, \rho_1, \rho_2, \rho_3 \rangle = D_{sk_{server}}(\gamma)$
		abort IF $\beta \neq b$
		abort IF $\Upsilon \neq \mathsf{NULL} \wedge st \neq \vartheta'$
		$\sigma = (\mathsf{encode}(m, r))^{d_2}\ mod\ N$
		$\vartheta \in_R \{0,1\}^k$
		$\theta = \vartheta \oplus \rho_1,\ \eta = \sigma \oplus \rho_2,\ \varphi = MAC_{\rho_3}(\langle \theta, \eta \rangle)$
		$count = count + 1$
		$\mathsf{append}(\tau, uid, count, \vartheta, m, r)$
	(θ, η, φ) $\longleftarrow$	
abort IF $MAC_{\rho_3}(\langle \theta, \eta \rangle) \neq \varphi,\ \sigma = \eta \oplus \rho_2,\ d_1 = f(v, pwd)$		
$s = \sigma \cdot (\mathsf{encode}(m, r))^{d_1}\ mod\ N$		
abort IF $s^e \neq \mathsf{encode}(m, r)$		
$st = \theta \oplus \rho_1,\ ct = ct + 1,\ token' = (ct, st, v, a, \tau, e, N, pk_{server})$		
erase $\beta,\ d_1,\ \rho_1,\ \rho_2,\ \rho_3,\ \theta,\ \eta,\ \varphi,\ \vartheta,\ token$		

Fig. 1. Building block: a single-server scheme with stateful soft-tokens

server will query its database to get $b = H_1(pwd)$ from $\langle a, b, uid, d_2, N \rangle = D_{sk_{server}}(\tau)$, where τ corresponds to *uid*. Note that any secure password protocol (e.g., [8, 4, 11, 22]) can be used for this purpose.

4.1 Discussions

On 3-factor authentication. Previous key-split schemes (such as [23]) employ a 2-factor authentication mechanism based on a password and a soft-token. Whereas, we employ a 3-factor authentication mechanism so that a signature is produced when a request

1. presents a valid password (i.e., what you know),
2. is initiated from a party having access to the user's soft-token (i.e., what you have), and
3. presents a valid fresh one-time secret (i.e., whether you *always* have access to the soft-token).

Indeed, the last factor helps achieve the newly introduced compromise detection and the enhanced abuse prevention (see Section 4.2). Note that the newly offered compromise detection is not fulfilled via an out-of-band intrusion detection system, which however may be deployed to implement another layer of protection.

On atomicity of the transactions. We assumed that atomicity of the transactions is ensured. This may be problematic when, for example, the servers are under a DDoS attack. This issue is addressed via another layer of assurance for synchronization in Section 5.

On light-weight key disabling. Allowing a user to disable its private key via a standard username/password authentication has the advantage that a user does not have to resort to its soft-token, which may not be available (e.g., when the user's device is stolen). Although this convenience seemingly gives an adversary the chance to impose denial-of-service attack (i.e., the adversary can request the server to disable the user's private key), we argue that there are no sever consequences.

1. An adversary knows nothing about a user's token or password. In this case the adversary can conduct an on-line dictionary attack against the user's password. This is sever if the on-line dictionary attack can be launched by a software program. Fortunately, there exist some effective methods (e.g., [27]) to force the adversary to conduct a *manual* on-line dictionary attack. It is likely that no adversary would conduct a manual on-line dictionary attack.
2. An adversary knows a user's token but not password. In this case, it is seemingly more attractive for the adversary to manage to get the user's password so that he can produce signatures (rather than disable its private key). Moreover, the user might also have to disable its private key once the user realized that its soft-token has been compromised.
3. An adversary knows a user's password but not token. In this case, the adversary can always disable the user's private key. We argue that this is no severe consequence because the user may also have to disable its private key once the user realized that its password has been compromised. Moreover, it might be unlikely that an adversary knows a user's password but not the user's token because the adversary could typically have had access to a user's computer (and token thereof). This is equivalent to the next case.
4. An adversary knows a user's token and password. This means that the adversary is already able to produce signatures by contacting the server, which is perhaps more attractive than to conduct a denial-of-service attack by disabling the victim user's private key.

4.2 Analysis

Our scheme does not incur any significant extra complexity, when compared with the starting-point scheme in [23]. Specifically, a soft-token keeps some state information (e.g., 160 bits), and a server keeps some state information linear to the number of users (which can be easily mitigated by letting the server use a pseudorandom function). Moreover, no extra exponentiations are imposed on a user or a server. Below we analyze the security properties.

Proposition 1. *The single server scheme implements* some *of the requirements specified in Section 3 (the others will be fulfilled in the full-fledged scheme via another layer of protection).*

* Abuse prevention. *This is analyzed in Theorem 1-5. Note that the first four theorems as well as their proofs are extended from [23], and Theorem 5 has no counterpart in [23].*

* Compromise detection. *Suppose atomicity of the transactions and integrity of the server are guaranteed. Lack of synchronization means that either an adversary had succeeded in impersonating the user, or the token had been tampered with. Note that this assurance is not offered in the original scheme in [23].*

* Immediate revocation. *This is true since the request for disabling a private key is authenticated by pwd, whereas uid is also remembered by the user. Note that this assurance is not offered in the original scheme in [23].*

In order to prove the abuse prevention, we introduce the following formal security model. Denote D-RSA$[\mathcal{E}, \mathbb{D}]$ the real-world single-server signing system based on an encryption scheme $\mathcal{E}$ for the server and dictionary $\mathbb{D}$. An adversary is given $\langle e, N\rangle$ where $(\langle e, N\rangle, \langle d, N, \phi(N)\rangle) \longleftarrow GEN_{RSA}(1^\lambda)$, the public data generated in the initialization procedure, and certain secret data of the user and/or server (depending on the type of the adversary). The goal of the adversary is to forge RSA signatures with respect to $\langle e, N\rangle$. The adversary is allowed to have the following types of oracle queries:

1. start(m) – This results in a user to initiate the protocol. The oracle may execute according to the protocol, maintain state as appropriate (i.e., there is an implicit notion of sessions), and return (γ, τ, δ).
2. serve(γ, τ, δ) – This represents the receipt of a message ostensively from the user. The oracle may execute according to the protocol to return (θ, η, φ).

3. $\mathsf{finish}(\theta, \eta, \varphi)$ – This represents the receipt of a response ostensively from the server. The oracle may execute according to the protocol to return a valid signature.

An adversary of type $\mathcal{ADV}\{server, pwd\}$, $\mathcal{ADV}\{token, server\}$, or $\mathcal{ADV}\{token\}$ succeeds in breaking the scheme if it can output a valid signature $\langle r, s\rangle$ on message m and there was no $\mathsf{start}(m)$ query. An adversary of type $\mathcal{ADV}\{token, pwd\}$ succeeds in breaking the scheme if it can output a valid signature $\langle r, s\rangle$ on message m and there was no $\mathsf{serve}(\gamma, \tau, \delta)$ query, where $\langle m, *, *, *, *, *, *\rangle = D_{sk_{server}}(\gamma)$. An adversary of type $\mathcal{ADV}\{\neg token^{(i)}, pwd\}$ succeeds in breaking the scheme if it can output a valid signature $\langle r, s\rangle$ on message m after the user finishes the i^{th} transaction and before the user initiates the $(i+1)^{th}$ transaction, and there was no $\mathsf{serve}(\gamma, \tau, \delta)$ query such that $\langle m, *, *, *, *, *, *\rangle = D_{sk_{server}}(\gamma)$.

Denote by q_{user} the number of $\mathsf{start}(\cdot)$ queries to the user, q_{server} the number of $\mathsf{serve}(\cdot,\cdot,\cdot)$ queries to the server. Let q_h and q_f be the number of queries to the random oracles h and f, respectively. Let q_o be the number of other oracle queries not counted above. Let $\bar{q} = (q_{user}, q_{server}, q_h, q_f, q_o)$. We also denote by $|\bar{q}| = q_{user} + q_{server} + q_h + q_f + q_o$ as the total number of oracle queries.

We say an adversary $(\bar{q}, \varepsilon)$-breaks D-RSA if it makes $\bar{q}$ oracle queries (of the respective types and to the respective oracles) and succeeds with probability at least ε. In the following, "$\approx$" means equality within negligible factors.

We say an attacker $\mathcal{A}$ (q, ε)-breaks $\mathcal{E}$ if the attacker makes q queries to the decryption oracle and $2 \cdot \Pr[\mathcal{A} \text{ succeeds}] - 1 \geq \varepsilon$, which implies $\Pr[\mathcal{A} \text{ outputs } 0|b=0] - \Pr[\mathcal{A} \text{ outputs } 0|b=1] \geq \varepsilon$. Note that if $\mathcal{E}$ uses random oracles, the oracles may be queried by the attacker along with the encryption oracle.

We say a forger (q, ε)-breaks a signature scheme if it makes q queries and succeeds with probability at least ε. Note that if $\mathcal{S}$ uses random oracles, the oracles may be queried by the forger along with the signature oracle.

Theorem 1. *Suppose $\{f_v\}$ is a pseudorandom function family. If an adversary $\mathcal{F}$ of type $\mathcal{ADV}\{server, pwd\}$ can $(\bar{q}, \varepsilon)$-break D-RSA$[\mathcal{E}, \mathbb{D}]$ system, then there exists a forger $\mathcal{F}^*$ able to (q_{user}, ε')-break the underlying RSA signature scheme, where $\varepsilon' \approx \varepsilon$.*

Proof. (sketch) Suppose $\mathcal{F}^*$ is given a RSA public key $\langle e, N\rangle$. The simulation goes as follows.

* $\mathcal{F}^*$ generates a pair of public and private keys $(pk_{server}, sk_{server})$ on behalf of the server, and chooses a password $pwd \in_R \mathbb{D}$ on behalf of the user. It generates ct, st, uid, v, a, b, $d_1 = f(v, pwd)$ as in the initialization protocol. However, it sets $d_2 \in_R Z_N^*$. It also generates $\tau = E_{pk_{server}}(\langle a, b, uid, d_2, N\rangle)$ and outputs $token = (ct, st, v, a, \tau, e, N, pk_{server})$. Finally, $\mathcal{F}^*$ gives $pk_{user} = \langle e, N\rangle$, pk_{server}, sk_{server}, and pwd to $\mathcal{F}$.
* $\mathcal{F}^*$ responds to a $\mathsf{start}(m)$ query by querying the RSA signature oracle to get $sig = \langle r, s\rangle$ and executing according to the protocol, except using r returned by the signature oracle. Finally, it returns (γ, τ, δ).
* $\mathcal{F}^*$ responds to a $\mathsf{serve}(\gamma, \tau, \delta)$ query as in a real execution by aborting or returning (θ, η, φ).
* $\mathcal{F}^*$ responds to a $\mathsf{finish}(\theta, \eta, \varphi)$ query corresponding to a $\mathsf{start}(m)$ query by checking if $MAC_{\rho_3}(\langle\theta, \eta\rangle) = \varphi$, computing $\sigma = \eta \oplus \rho_2$ and checking if $\sigma = (\mathsf{encode}(m, r))^{d_2} \ mod\ N$, where ρ_2 and ρ_3 are from the start (m) query. If any of the two conditions is not satisfied, $\mathcal{F}^*$ aborts the user as in a real execution; otherwise, $\mathcal{F}^*$ returns $sig = \langle r, s\rangle$ obtained from the signature oracle and generates a new token as in a real execution.

Denote by ε' the probability that $\mathcal{F}$ succeeds in forging a signature in the above simulation. Consider an experiment in which f_v is replaced with a random function. Denote by ε'' the probability that $\mathcal{F}$ succeeds in forging a signature in the experiment. Due to the statistical indistinguishability between the simulation and the experiment, we have $\varepsilon' \approx \varepsilon''$. Due to the pseudorandomness of f_v, we have $\varepsilon \approx \varepsilon''$. Therefore, $\mathcal{F}^*$ succeeds in forging a signature with probability at least $\varepsilon' \approx \varepsilon$ after q_{user} queries to the signature oracle. □

Theorem 2. *Let h and f be random oracles. If an adversary $\mathcal{F}$ of type $\mathcal{ADV}\{token, server\}$ can $(\bar{q}, \varepsilon)$-break D-RSA$[\mathcal{E}, \mathbb{D}]$ system, then there exists a forger $\mathcal{F}^*$ able to (q_{user}, ε')-break the underlying RSA signature scheme, where $\varepsilon' \approx \varepsilon - \frac{q_h+q_f}{|\mathbb{D}|}$.*

Proof. (sketch) Suppose $\mathcal{F}^*$ is given a RSA public key $\langle e, N\rangle$. The simulation goes as follows.

* $\mathcal{F}^*$ generates a pair of public and private keys $(pk_{server}, sk_{server})$ on behalf of the server, and chooses a password $pwd \in_R \mathbb{D}$ on behalf of the user. It generates ct, st, uid, v, a, b, $d_1 = f(v, pwd)$ as in the initialization protocol. However, it generates d_2 by choosing $d_2 \in_R Z_N^*$. It generates $\tau = E_{pk_{server}}(\langle a, b, uid, d_2, N\rangle)$ and outputs $token = (ct, st, v, a, \tau, e, N, pk_{server})$. Finally, $\mathcal{F}^*$ gives $token$ and sk_{server} to $\mathcal{F}$.

* $\mathcal{F}^*$ responds to a $\mathsf{start}(m)$ query by querying the signature oracle to get $sig = \langle r, s\rangle$ and executing according to the protocol, except using r returned by the signature oracle. Finally, it returns (γ, τ, δ).

* $\mathcal{F}^*$ responds to a $\mathsf{serve}(\gamma, \tau, \delta)$ query as in a real execution by aborting or returning (θ, η, φ).

* $\mathcal{F}^*$ responds to a $\mathsf{finish}(\theta, \eta, \varphi)$ query corresponding to a $\mathsf{start}(m)$ query by checking if $MAC_{\rho_3}(\langle\theta, \eta\rangle) = \varphi$, computing $\sigma = \eta \oplus \rho_2$ and checking if $\sigma = (\mathsf{encode}(m, r))^{d_2}\ mod\ N$, where ρ_2 and ρ_3 are from the start (m) query. If any of the two conditions is not satisfied, $\mathcal{F}^*$ aborts the user as in a real execution; otherwise, $\mathcal{F}^*$ returns $sig = \langle r, s\rangle$ obtained from the signature oracle, generates a new $token$ as in a real execution. The new $token$ is also given to the adversary.

* $\mathcal{F}^*$ responds to a $h(pwd^*)$ or $f(v^*, pwd^*)$ query as a normal random oracle, except that $\mathcal{F}^*$ aborts the simulation if $pwd^* = pwd$ in a $h(\cdot)$ query, or $v^* = v$ and $pwd^* = pwd$ in a $f(\cdot, \cdot)$ query.

The simulation is indistinguishable from the real-world execution, unless $\mathcal{F}$ hits pwd, which happens with probability negligibly more than $\frac{q_h+q_f}{|\mathbb{D}|}$. Therefore, Pr[$\mathcal{F}^*$ succeeds] = Pr[$\mathcal{F}$ succeeds $\wedge$ $\neg$ $\mathcal{F}$ hits pwd] $\geq \varepsilon - \frac{q_h+q_f}{|\mathbb{D}|}$. □

Theorem 3. *Suppose H_1 has a negligible probability of collision over $\mathbb{D}$. If an adversary $\mathcal{A}$ of type $\mathcal{ADV}\{token\}$ can $(\bar{q}, \varepsilon)$-break the D-RSA$[\mathcal{E}, \mathbb{D}]$ system for $\varepsilon = \frac{q_{server}}{|\mathbb{D}|} + \psi$, then there exists either a forger $\mathcal{F}^*$ able to (q_{user}, ε')-break the underlying RSA signature scheme for $\varepsilon' \approx \frac{\psi}{2}$, or an attacker $\mathcal{A}^*$ able to $(2q_{server}, \varepsilon'')$-break $\mathcal{E}$ for $\varepsilon'' \approx \frac{\psi}{2(1+q_{user})}$.*

Proof. (sketch) We construct a simulation in which τ and all the γ's are the encryptions of 0-string of appropriate length. Define that an adversary $\mathcal{A}$ *wins* in the simulation if he succeeds in forging a signature, or hits the pwd. By "$\mathcal{A}$ hits the pwd" we mean that, when $\mathcal{A}$ imposes a $\mathsf{serve}(\gamma, \tau^*, \delta)$ query, one of the following happens.

* In the case that $\tau^* = \tau$ yet γ is not from a $\mathsf{start}(m)$ query, the following conditions are satisfied simultaneously. Denote $\langle m, r, \beta, st, \rho_1, \rho_2, \rho_3\rangle = D_{sk_{server}}(\gamma)$ and $\Upsilon = \mathsf{last}(\tau, uid, count, \vartheta', m', r')$.
  - $\delta = MAC_a(\langle\gamma, \tau^*\rangle)$, where a is the key chosen in the initialization protocol.
  - $\beta = b$, where b is from the initialization protocol.
  - $st = \vartheta'$ if $\Upsilon \neq \mathsf{NULL}$.

* In the case that $\tau^* \neq \tau$ yet γ is from a $\mathsf{start}(m)$ query, the following conditions are satisfied simultaneously. Denote $\langle a^*, b^*, uid^*, d_2^*, N^*\rangle = D_{sk_{server}}(\tau^*)$ and $\Upsilon = \mathsf{last}(\tau^*, uid^*, count^*, \vartheta'', m'', r'')$.
  - $\delta = MAC_{a^*}(\langle\gamma, \tau^*\rangle)$.
  - $b^* = \beta$, where β is from the $\mathsf{start}(m)$ query.
  - $st = \vartheta''$ if $\Upsilon \neq \mathsf{NULL}$, where st is from the $\mathsf{start}(m)$ query.

If Pr[$\mathcal{A}$ wins] $\geq \frac{q_{server}}{|\mathbb{D}|} + \frac{\psi}{2}$, then there exists $\mathcal{F}^*$ able to (q_{user}, ε')-break the underlying RSA signature scheme for $\varepsilon' \approx \frac{\psi}{2}$; otherwise, there exists $\mathcal{A}^*$ able to $(2q_{server}, \varepsilon'')$-break $\mathcal{E}$, where $\varepsilon'' \approx \frac{\psi}{2(1+q_{user})}$.

Now, we present the detailed simulation. Suppose $\mathcal{F}^*$ is given a RSA public key $\langle e, N\rangle$.

* $\mathcal{F}^*$ generates a pair of public and private keys $(pk_{server}, sk_{server})$ on behalf of the server, and chooses a password $pwd \in_R \mathbb{D}$ on behalf of the user. It generates ct, st, uid, v, a, b, $d_1 = f(v, pwd)$ as in the initialization protocol. However, it generates d_2 by choosing $d_2 \in_R Z_N^*$, sets $\tau = E_{pk_{server}}(\langle 0^{3k+2\lambda} \rangle)$, and outputs $token = (ct, st, v, a, \tau, e, N, pk_{server})$. Finally, $\mathcal{F}^*$ gives $token$ to $\mathcal{F}$.
* $\mathcal{F}^*$ responds to a $\mathsf{start}(m)$ query by querying the signature oracle to get $sig = \langle r, s \rangle$ and executing according to the protocol. However, it returns (γ, τ, δ) where $\gamma = E_{pk_{server}}(0^{|m|+len_{pad}+4k+\lambda})$.
* $\mathcal{F}^*$ responds to a $\mathsf{serve}(\gamma, \tau^*, \delta)$ query as follows.
  1. (γ, τ^*, δ) is from a $\mathsf{start}(m)$ query. It returns (θ, η, φ) by computing $\theta = \rho_1 \oplus \vartheta$, $\eta = \rho_2 \oplus ((\mathsf{encode}(m, r))^{d_2}\ mod\ N)$, and $\varphi = MAC_{\rho_3}(\langle \theta, \eta \rangle)$, where m, r, ρ_1, ρ_2, and ρ_3 are from the $\mathsf{start}(m)$ query, d_2 is from the simulated initialization, and $\vartheta \in_R \{0,1\}^k$.
  2. γ is from a $\mathsf{start}(m)$ query, but $\tau^* \neq \tau$. Denote $\langle a^*, b^*, uid^*, d_2^*, N^* \rangle = DEC_{sk_{server}}(\tau^*)$ and $\Upsilon = \mathsf{last}(\tau^*, uid^*, count^*, \vartheta'', m'', r'')$. It aborts the server if $\delta \neq MAC_{a^*}(\langle \gamma, \tau^* \rangle)$ as in a real execution. It *aborts the simulation* if either $b^* = \beta$ in the case of $\Upsilon = \mathsf{NULL}$, or the predicate $b^* = \beta \wedge st = \vartheta''$ is satisfied in the case of $\Upsilon \neq \mathsf{NULL}$, where β and st are from the $\mathsf{start}(m)$ query. Otherwise, it aborts the server as in a real execution since either $b^* \neq \beta$ in the case of $\Upsilon = \mathsf{NULL}$, or the predicate $b^* = \beta \wedge st = \vartheta''$ is not satisfied in the case of $\Upsilon \neq \mathsf{NULL}$.
  3. γ and τ are from a $\mathsf{start}(m)$ query, but not δ. It aborts the server.
  4. γ is not from a $\mathsf{start}(m)$ query, but $\tau^* = \tau$. Denote $\langle m, r, \beta, st, \rho_1, \rho_2, \rho_3 \rangle = D_{sk_{server}}(\gamma)$ and $\Upsilon = \mathsf{last}(\tau, uid, count, \vartheta', m', r')$. It aborts the server if $\delta \neq MAC_a(\langle \gamma, \tau^* \rangle)$, where a is from the simulated initialization. It *aborts the simulation* if either $\beta = b$ in the case of $\Upsilon = \mathsf{NULL}$, or the predicate $\beta = b \wedge st = \vartheta'$ is satisfied in the case of $\Upsilon \neq \mathsf{NULL}$. Otherwise, it aborts the server as in a real execution.
  5. Neither γ is from a $\mathsf{start}(m)$ query, nor is $\tau^* = \tau$. In this case, let the server behave according to the protocol.
* $\mathcal{F}^*$ responds to a $\mathsf{finish}(\theta, \eta, \varphi)$ query corresponding to a $\mathsf{start}(m)$ query by checking if $MAC_{\rho_3}(\langle \theta, \eta \rangle) = \varphi$, computing $\sigma = \eta \oplus \rho_2$ and checking if $\sigma = (\mathsf{encode}(m, r))^{d_2}\ mod\ N$, where m, r, ρ_2 and ρ_3 are chosen in the $\mathsf{start}(m)$ query. If any of the two conditions is not satisfied, $\mathcal{F}^*$ aborts the user as in a real execution; otherwise, $\mathcal{F}^*$ returns $sig = \langle r, s \rangle$ obtained from the signature oracle and generates a new $token$. The new $token$ is also given to the adversary.

The probability that $\mathcal{A}$ hits pwd is negligibly more than $\frac{q_{server}}{|\mathbb{D}|}$, disregarding the negligible probability of collision. Therefore, if $\mathcal{A}$ wins in the simulation with probability at least $\frac{q_{server}}{|\mathbb{D}|} + \frac{\psi}{2}$, $\mathcal{F}^*$ can (q_{user}, ε')-break RSA with probability at least $\varepsilon' \approx \frac{\psi}{2}$, since $\Pr[\mathcal{F}^* \text{ succeeds}] = \Pr[\mathcal{A} \text{ wins} \wedge \neg\mathcal{A} \text{ hits } pwd] \geq \frac{\psi}{2}$.

Now, suppose $\mathcal{A}$ wins in the simulation with probability less than $\frac{q_{server}}{|\mathbb{D}|} + \frac{\psi}{2}$. Suppose $\mathcal{A}^*$ is given a public key pk'. $\mathcal{A}^*$ can *perfectly* simulate the real-world system as follows. It generates the pair of public and private keys on behalf of the user. It will also need to have access to the decryption oracle, but at most $2q_{server}$ times since the decryptions of ciphertexts generated by $\mathcal{A}^*$ have already been known to $\mathcal{A}^*$. According to the assumption, $\mathcal{A}$ succeeds in forging in this simulation with probability at least $\frac{q_{server}}{|\mathbb{D}|} + \psi$.

Consider the same simulation, except that τ and all γ's generated by $\mathcal{A}^*$ are all encryptions of 0-string of appropriate length. This simulation differs from the simulation by $\mathcal{F}^*$ only in that it does not abort in the case that $\mathcal{A}$ hits the password. Therefore, the probability that $\mathcal{A}$ succeeds in forging in this simulation is at most $\frac{q_{server}}{|\mathbb{D}|} + \frac{\psi}{2}$.

In order to use the hybrid argument, define experiment $\mathcal{EXPT}_j$, where $j \in_R \{0, \cdot\cdot, q_{user} + 1\}$, corresponding to the situation that the first j ciphertexts generated by $\mathcal{A}^*$ are of normal messages, and the rest are encryptions of 0-string of appropriate length. Let p_j be the probability that $\mathcal{A}$ succeeds in forging in $\mathcal{EXPT}_j$. Thus, $p_0 \leq \frac{q_{server}}{|\mathbb{D}|} + \frac{\psi}{2}$ and $p_{q_{user}+1} > \frac{q_{server}}{|\mathbb{D}|} + \psi$. For $i \in_R \{1, \cdot\cdot, q_{user} + 1\}$, the probability that $\mathcal{A}$ can distinguish the encryption of a normal message from the encryption of

0-string of the same length is $\Pr[\mathcal{A} \text{ breaks } \mathcal{E}] = \frac{1}{q_{user}+1}\sum_{j=1}^{q_{user}+1}(p_j - p_{j-1}) \geq \frac{\psi}{2(1+q_{user})}$. So, let $\mathcal{A}^*$ simply choose $i \in_R \{1, \cdot\cdot, q_{user}+1\}$, and run experiment $\mathcal{EXPT}_i$. In order to embed the challenge into the i^{th} encryption, let $\mathcal{A}^*$ send a pair of messages (X_0, X_1) to the test oracle with respect to $\mathcal{E}$, where X_0 is a normal message and X_1 is 0-string of the same length. The challenge ciphertext received from the test oracle is embedded as the i^{th} encryption. Now, $\mathcal{A}^*$ outputs 0 (meaning that the encryption is of X_0) if $\mathcal{F}$ succeeds in forging in $\mathcal{EXPT}_i$, and a random bit otherwise. Therefore, $\mathcal{A}^*$ can $(2q_{server}, \frac{\psi}{2(1+q_{user})})$-break $\mathcal{E}$. □

Theorem 4. *Suppose the underlying RSA signature scheme is deterministic. If an adversary $\mathcal{A}$ of type $\mathcal{ADV}\{token, pwd\}$ can $(\bar{q}, \varepsilon)$-break the D-RSA$[\mathcal{E}, \mathbb{D}]$ system, then there exists either a forger $\mathcal{F}^*$ able to $(q_{server}, \varepsilon')$-break the underlying RSA signature scheme for $\varepsilon' \approx \frac{\varepsilon}{2}$, or an attacker $\mathcal{A}^*$ able to $(2q_{server}, \varepsilon'')$ -break $\mathcal{E}$ for $\varepsilon'' \approx \frac{\varepsilon}{2(1+q_{user})}$.*

Proof. (sketch) We construct a simulation in which τ and the γ's are the encryptions of 0-string of appropriate length. An adversary $\mathcal{A}$ succeeds in the simulation, if he can forge a signature. We use a new query $\mathsf{coordinate}(token)$ to model the capability that an adversary can maintain the consistence of the state information between the user side and the server side.

Now, we present the detailed simulation. Suppose $\mathcal{F}^*$ is given a RSA public key $\langle e, N\rangle$.

* $\mathcal{F}^*$ generates a pair of public and private keys $(pk_{server}, sk_{server})$ on behalf of the server, and chooses a password $pwd \in_R \mathbb{D}$ on behalf of the user. It generates ct, st, uid, v, a, b, $d_1 = f(v, pwd)$ as in the initialization protocol. However, it generates d_2 by choosing $d_2 \in_R Z_N^*$. It encapsulates the data into $\tau = E_{pk_{server}}(\langle 0^{3k+2\lambda}\rangle)$ and outputs $token = (ct, st, v, a, \tau, e, N, pk_{server})$. Finally, $\mathcal{F}^*$ gives $\mathcal{A}$ the $token$ and pwd.
* $\mathcal{F}^*$ responds to a $\mathsf{start}(m)$ query as in a real execution, except that it returns (γ, τ, δ) where $\gamma = E_{pk_{server}}(0^{|m|+len_{pad}+4k+\lambda})$.
* $\mathcal{F}^*$ responds to a $\mathsf{serve}(\gamma, \tau^*, \delta)$ query as follows.
  1. (γ, τ^*, δ) is from a $\mathsf{start}(m)$ query. Query the signature oracle to obtain signature $\langle r, s\rangle$. It returns (θ, η, φ) by computing $\theta = \rho_1 \oplus \vartheta$, $\eta = \rho_2 \oplus (s/(\mathsf{encode}(m, r))^{d_1} \ mod\ N)$, and $\varphi = MAC_{\rho_3}(\langle\theta, \eta\rangle)$, where m, r, ρ_1, ρ_2, and $\rho_3$are from the $\mathsf{start}(m)$ query, d_1 is from the simulated initialization, and $\vartheta \in_R \{0,1\}^k$.
  2. γ is from a $\mathsf{start}(m)$ query, but $\tau^* \neq \tau$. It executes according to the protocol, but using m, r, β, ρ_1, ρ_2, and ρ_3 from the $\mathsf{start}(m)$ query.
  3. γ and τ are from a $\mathsf{start}(m)$ query, but not δ. It aborts the server as in a real execution.
  4. γ is not from a $\mathsf{start}(m)$ query, but $\tau^* = \tau$. Denote $\langle m, r, \beta, st, \rho_1, \rho_2, \rho_3\rangle = DEC_{sk_{server}}(\gamma)$, and $\Upsilon = \mathsf{last}(\tau, uid, count, \vartheta', m', r')$. It aborts the server if $\delta \neq MAC_a(\langle\gamma, \tau^*\rangle)$ where a is chosen in the simulated initialization protocol. It aborts the server if $\beta \neq b$ in the case of $\Upsilon = \mathsf{NULL}$, or the predicate $\beta = b \wedge st = \vartheta'$ is not satisfied in the case of $\Upsilon \neq \mathsf{NULL}$. If nothing goes wrong, it queries the signature oracle to obtain the signature $sig = \langle r, s\rangle$, and returns (θ, η, φ), where $\theta = \rho_1 \oplus \vartheta$, $\eta = \rho_2 \oplus (s/(\mathsf{encode}(m, r))^{d_1} \ mod\ N)$, $\varphi = MAC_{\rho_3}(\langle\theta, \eta\rangle)$, and d_1 is from the simulated initialization protocol.
  5. Neither γ is from a $\mathsf{start}(m)$ query, nor is $\tau^* = \tau$. In this case, let the server behave as in a real execution.
* $\mathcal{F}^*$ responds to a $\mathsf{finish}(\theta, \eta, \varphi)$ query corresponding to a $\mathsf{start}(m)$ query by checking if $MAC_{\rho_3}(\langle\theta, \eta\rangle) = \varphi$, computing $\sigma = \eta \oplus \rho_2$ and checking if $\sigma = s/(\mathsf{encode}(m, r))^{d_1} \ mod\ N$, where m, r, ρ_2 and ρ_3 are chosen in the $\mathsf{start}(m)$ query, and d_1 is from the simulated initialization protocol. If any of the two conditions is not satisfied, $\mathcal{F}^*$ aborts the user; otherwise, $\mathcal{F}^*$ returns $sig = \langle r, s\rangle$ obtained from the $\mathsf{serve}(\gamma, \tau, \delta)$ query and generates a new $token$ that is also given to the adversary.
* $\mathcal{F}^*$ responds to a $\mathsf{coordinate}(token)$ query by replacing the token at the user side with $token$.

If $\mathcal{A}$ succeeds in forging a signature in the simulation with probability at least $\frac{\varepsilon}{2}$, $\mathcal{F}^*$ can $(q_{server}, \varepsilon')$-break RSA with probability at least $\varepsilon' \approx \frac{\varepsilon}{2}$.

Now, suppose the adversary $\mathcal{A}$ succeeds in forging a signature in the above simulation with probability less than $\frac{\varepsilon}{2}$. We construct $\mathcal{A}^*$ that $(2q_{server}, \varepsilon'')$-breaks the encryption scheme $\mathcal{E}$, where $\varepsilon'' \approx \frac{\varepsilon}{2(1+q_{user})}$. Suppose $\mathcal{A}^*$ is given a public key pk'. $\mathcal{A}^*$ can *perfectly* simulate the real world D-RSA as follows. It generates the pair of public and private keys on behalf of the user. It will also need to have access to the decryption oracle, but at most $2q_{server}$ times since the decryptions of ciphertexts generated by $\mathcal{A}^*$ have already been known to $\mathcal{A}^*$. According to the assumption, $\mathcal{A}$ succeeds in forging in this simulation with probability at least ε.

Now consider the same simulation, except that τ and all the γ's generated by $\mathcal{A}^*$ are encryptions of 0-string of appropriate length. This simulation is equivalent to the simulation of $\mathcal{F}^*$. Therefore, $\mathcal{A}$ succeeds in forging in this simulation with probability at most $\frac{\varepsilon}{2}$. By a similar argument as in Theorem 3, we know that $\mathcal{A}^*$ can $(2q_{server}, \frac{\varepsilon}{2(1+q_{user})})$-break $\mathcal{E}$. □

Theorem 5. *Suppose the underlying RSA signature scheme is deterministic. If an adversary $\mathcal{A}$ of type $\mathcal{ADV}\{\neg token^{(i)}, pwd\}$ can $(\bar{q}, \varepsilon)$-break the D-RSA$[\mathcal{E}, \mathbb{D}]$ system, then there exists either a forger $\mathcal{F}^*$ able to $(q_{server}, \varepsilon')$-break the underlying RSA signature scheme for $\varepsilon' \approx \frac{\varepsilon}{2q_{user}}$, or an attacker $\mathcal{A}^*$ able to $(2q_{server}, \varepsilon'')$-break $\mathcal{E}$ for $\varepsilon'' \approx \frac{\varepsilon}{4q_{user}}$.*

Proof. (sketch) Suppose there exists i, $1 \le i \le q_{user}$, such that $\mathcal{A}$ outputs a valid signature for a new message with probability at least ε after the user finishes the i^{th} transaction and before the user initiates the $(i+1)^{th}$ transaction. Consider an algorithm $\mathcal{F}^{**}$ that is the same as $\mathcal{F}^*$ in Theorem 4, except that:

* Let τ be the encryption of 0-string of appropriate length, the first $i-1$ γ's be the encryptions of normal messages, and the i^{th} γ be the encryption of 0-string of appropriate length.
* A token is sent to $\mathcal{A}$ only when it issues a get() query, which is not allowed for the i^{th} token generated by the user in the i^{th} transaction. Also, $\mathcal{A}$ can maintain the consistency between the user side and the server side by issuing a coordinate(*token*) query.

Given the simulation generated by $\mathcal{F}^{**}$, if $\Pr[\mathcal{A}$ succeeds$] \ge \frac{\varepsilon}{2}$, then it is clear that there exists $\mathcal{F}^*$ able to $(q_{server}, \varepsilon')$-break the underlying RSA signature scheme, where $\varepsilon' \approx \frac{\varepsilon}{2q_{user}}$; otherwise, there exists $\mathcal{A}^*$ able to $(2q_{server}, \varepsilon'')$-break $\mathcal{E}$, where $\varepsilon'' \approx \frac{\varepsilon}{4q_{user}}$.

Consider an algorithm $\mathcal{A}^{**}$ that is given a public key pk'. Now, $\mathcal{A}^{**}$ can *perfectly* simulate the real-world system as follows. It generates the pair of public and private keys on behalf of the user. It need have access to the decryption oracle, but at most $2q_{server}$ times. Note that $\mathcal{A}$ is given a newly generated token only when it issues a get() query, and that no get() query is allowed after the user finishes the i^{th} transaction and before the user initiates the $(i+1)^{th}$ transaction. Therefore, $\mathcal{A}$ succeeds in forging with probability at least ε.

Now consider the same simulation, except that τ is the encryption of 0-string of appropriate length, the first $i-1$ γ's are the encryptions of normal messages, and the i^{th} γ is the encryption of 0-string of appropriate length. This simulation is equivalent to the simulation of $\mathcal{F}^{**}$. Therefore, $\Pr[\mathcal{A}$ succeeds$] < \frac{\varepsilon}{2}$.

A standard hybrid argument shows that $\mathcal{A}^{**}$ can $(2q_{server}, \frac{\varepsilon}{4})$-break $\mathcal{E}$, which means that there exists $\mathcal{A}^*$ able to $(2q_{server}, \frac{\varepsilon}{4q_{user}})$-break $\mathcal{E}$. □

5 Full-Fledged Scheme

Now we present the full-fledged scheme, which is built on top of the building-block discussed in the previous section. The basic idea underlying the scheme is the following. In order to achieve compromise confinement, we augment $b = H_1(pwd)$ to $b = H_1(c, pwd)$, where c is a cryptographic secret. The motivation is to ensure that $b = H_1(c, pwd)$ itself does not leak any significant information of a password *pwd*. A drawback of this approach is that a user can not disable its private key using a standard username/password mechanism. Nevertheless, this can be resolved by letting a user choose two passwords, one for generating signatures and the other for disabling its key. Specifically, this can be done by further augmenting $\tau = E_{pk_{server}}(\langle a, b, uid, d_2, N\rangle)$ to $\tau = E_{pk_{server}}(\langle a, b, b^*, uid, d_2, N\rangle)$ such that $b = H_1(c, pwd)$ and $b^* = H_2(\pi)$, where password pwd is for generating signatures, and password π is for disabling the key.

Special care is also taken to address atomicity and availability issues. Suppose a server $server_i$ can not finish a transaction within a certain time interval. Then, it is reasonable to allow the user to contact another server. This flexibility complicates the facilitation of transaction atomicity. We resolve this issue by incorporating a simple "commit/rollback" mechanism. Moreover, a *commit* or *rollback* request should be authenticated. This can be done by augmenting $\gamma = E_{pk_{server}}(\langle m, r, \beta, st, \rho_1, \rho_2, \rho_3\rangle)$ to $\gamma = E_{pk_{server}}(\langle m, r, \beta, st, \rho_1, \rho_2, \rho_3, \rho_4, \rho_5\rangle)$, where $\rho_4 = H_3(\rho_4^*)$ is used for committing a transaction and $\rho_5 = H_3(\rho_5^*)$ is used for rollbacking a transaction.

The scheme has the following components.

Token initialization. Suppose $H_1 : \{0,1\}^* \longrightarrow \{0,1\}^k$, $H_2, H_3 : \{0,1\}^k \longrightarrow \{0,1\}^k$, and $f : \{0,1\}^* \rightarrow \{0,1\}^{\lambda+k}$ are appropriate hash and pseudorandom functions, respectively. A user chooses two passwords – pwd for generating signatures and π for disabling the private key, and a pair of public and private keys ($pk_{user} = \langle e, N\rangle; sk_{user} = \langle d, N, \phi(N)\rangle$). A user chooses n servers as its service providers, and each server has a public key $pk_{server,i}$, where $1 \le i \le n$. The initialization process proceeds as follows.

$$\begin{array}{lll}
uid & = & username \\
v_i & \in_R & \{0,1\}^k,\ 1 \le i \le n \\
a_i & \in_R & \{0,1\}^k,\ 1 \le i \le n \\
c_i & \in_R & \{0,1\}^k,\ 1 \le i \le n \\
b_i & = & H_1(c_i, pwd),\ 1 \le i \le n \\
b^* & = & H_2(\pi) \\
d_{1,i} & = & f(v_i, pwd),\ 1 \le i \le n \\
d_{2,i} & = & d - d_{1,i} \bmod \phi(N),\ 1 \le i \le n \\
\tau_i & = & E_{pk_{server,i}}(\langle a_i, b_i, b^*, uid, d_{2,i}, N\rangle),\ 1 \le i \le n \\
ct_i & = & 0,\ 1 \le i \le n \\
st_i & \in_R & \{0,1\}^k,\ 1 \le i \le n \\
token_i & = & (ct_i, st_i, v_i, a_i, c_i, \tau_i, e, N, pk_{server,i}),\ 1 \le i \le n
\end{array}$$

At the end of the initialization, the user erases all the other values and keeps n soft-tokens on its device.

Server databases. Server $server_i$, $1 \le i \le n$, maintains a database of $\Upsilon_i = (\tau, uid, count, \vartheta, m, r)$, which represents a transaction corresponding to $\tau = E_{pk_{server}}(\langle a, b, b^*, uid, d_2, N\rangle)$, where uid is obtained after receiving the first service request, $count$ is an incremental counter (with initialized value zero) maintained by the server. There are two types of operations regarding the database. First, $\mathsf{append}(\tau, uid, count, \vartheta, m, r)$ appends the tuple $(\tau, uid, count, \vartheta, m, r)$ into the database. Second, $\mathsf{last}(\tau, uid, count, \vartheta, m, r)$ returns either the tuple $(\tau, uid, count, \vartheta, m, r)$ corresponding to the *last* transaction corresponding to τ, or NULL meaning that the token corresponding to τ has never been used before.

Signing protocol. The protocol is depicted in Figure 2. The user first sends a request to a (randomly chosen) server. If the transaction can not be finished within certain time, the user contacts another server. Note that such a switching process can be made transparent to the user.

Key disabling protocol. In order to disable its private key, the user authenticates itself to each of the n servers by proving that it knows the password π corresponding to uid. This process is the same as in the underlying single server protocol, and can be made automatic via an appropriate software design (for ease of deployment).

```
USER                                                    SERVER_1, ..., SERVER_n

$token_j = (ct_j, st_j, v_j, a_j, \tau_j, e, N, pk_{server,j})$, $1 \le j \le n$
$trial = \phi$
REPEAT if a transaction cannot be finished within time $\Delta$
    choose $i \in \{1, 2, \cdots, n\} - trial$ according to certain policy (e.g., random)
    $trial = trial \cup \{i\}$

    user provides $token_i$ and $\gamma = E_{pk_{server,i}}(\langle m, r_i, \beta_i, st_i, \rho_{1,i}, \rho_{2,i}, \rho_{3,i}, \rho_{4,i}, \rho_{5,i} \rangle)$.
    The server executes as follows:
        If $st_i$ matches the last "committed" transaction, execute the single server protocol;
        if $st_i$ matches the last "pending" transaction,
            change its status as "committed" and execute as specified;
        otherwise, reject the request and inform "user compromised."

UNTIL there is a successful transaction using $token_j$
$\alpha = MAC_{\rho_{3,j}}(\text{“commit”}, \tau_j, \rho^*_{4,j})$
$token_j$ is updated
                    $\xrightarrow{(\text{“commit”}, \tau_j, \rho^*_{4,j}, \alpha)}$
                                                    $j^{th}$ server commits the transaction
$trial = trial - \{j\}$
FOR each $i \in trial$
    $\alpha = MAC_{\rho_{3,i}}(\text{“rollback”}, \tau_i, \rho^*_{5,i})$
                    $\xrightarrow{(\text{“rollback”}, \tau_i, \rho^*_{5,i}, \alpha)}$
                                                    $i^{th}$ server rollbacks the transaction
```

Fig. 2. The full-fledged scheme

5.1 Analysis and Discussion

Since transaction atomicity plays an important role in our system, we must show that it has no significant security consequence if atomicity is violated.

Proposition 2. *Suppose there is no system crash resulting in the loss of system state information. Suppose a server keeps a database of state information $(\tau, uid, count, \vartheta, m, r)$. Then there is no denial of service attack because of an out of synchronization between a user and a server.*

Proof. Recall that a server classifies transactions into two categories: "committed" and "pending". Note that a successfully rollbacked transaction is treated as a "committed" one, but corresponding to the last successfully committed transaction. This is so because that a rollbacked transaction can be viewed as one where no actions have been taken whatsoever. So, when a user sends a request with certain state information ϑ, there are three cases.

* ϑ matches the state information corresponding to a "committed" transaction $\mathcal{T}$. There are further two cases.
  1. $\mathcal{T}$ is the last "committed" transaction. The This is the normal case and the protocol proceeds as specified.

2. $\mathcal{T}$ is not the last “committed” transaction. The server simply rejects this request. In this case, the server could inform the user, perhaps via an out-of-band channel, that the user side might have been compromised. This is so because in our design a user updates its state information before sending a commitment request.

* ϑ matches the state information corresponding to a “pending” transaction $\mathcal{T}$. There are further two cases.
  1. $\mathcal{T}$ is the last “pending” transaction. This means that the server did not commit the transaction yet. Then the server can simply accept the request and change the “pending” transaction into a “committed” one.
  2. $\mathcal{T}$ is not the last “committed” transaction. This is impossible, because our design ensures that whenever a server accepts a request (and sends new state information to a user), it always treat the *last* transaction as committed.
* ϑ matches no state information in the server database at all. The server simply rejects the request. In this case, the server could inform the user, perhaps via an out-of-band channel, that the user side might have been compromised. This is so because in our design we let a server choose the state information.

□

The above proposition implies that we can treat all transactions as atomic. Now we are ready to look at the properties of the full-fledged scheme. We claim that the full-fledged scheme implements all of the goals specified in Section 3. Informally, we observe the following:

* Abuse prevention. This can be formally analyzed by extending the theorems in the underlying single server scheme.
* Compromise detection. Suppose atomicity of the transactions and integrity of the server are guaranteed. Lack of synchronization means either an adversary had successfully impersonated the user, or the token had been tampered with. In either case, the user needs to disable her private key.
* Immediate revocation. This is true since the request for disabling a private key is authenticated by π, whereas *uid* is also remembered by the user.
* Compromise confinement. Once it is known that a server has been compromised, only the users who suspects that their tokens have been compromised need to re-initialize their keys, whereas the others just need to erase their tokens corresponding to the compromised server (i.e., no need to re-initialize their private keys). We notice that when a user knows that its password for disabling its private key, namely π, may have been compromised after off-line dictionary attack — due to the compromise of a server, the user can, if desired, execute an appropriate password-change protocol (e.g., the one associated with the adopted password authentication scheme) to update π to some π'. Such a process would be much more light-weight than a process for updating private keys. Note that the password update process is not necessary from the perspective of the security of the user's signature scheme.
* Scalability. The system is scalable because the servers are decentralized, namely that the servers are operated by possibly many service providers.
* High availability. The system is highly available since a single server is sufficient for the users to generate signatures.

6 Conclusion and Future Work

We presented a scalable and secure cryptographic service, which has the following features: (1) it incorporates a 3-factor authentication mechanism, which facilitates compromise detection; (2) it supports immediate revocation of a cryptographic functionality in question; (3) the damage due to the compromise of a server is contained; (4) it is scalable and highly available.

Acknowledgement. We thank Philip MacKenzie for valuable feedback.

References

1. R. Anderson. Invited Talk at ACM CCS'97.
2. N. Asokan, G. Tsudik, and M Waidner. Server-Supported Signatures. Journal of Computer Security. 5(1), 1997.
3. M. Bellare and S. Miner. A forward-secure digital signature scheme. In Proc. Crypto'99.
4. M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated Key Exchange Secure against Dictionary Attacks. In Proc. Eurocrypt'00.
5. M.Bellare and P. Rogaway. Random Oracles Are Practical: A Paradigm for Designing Efficient Protocols. In Proc. ACM CCS'93, pp 62-73.
6. M. Bellare and P. Rogaway. Optimal asymmetric encryption – How to encrypt with RSA. In Proc. Eurocrypt'94.
7. M. Bellare and P. Rogaway. The Exact Security of Digital Signatures - How to Sign with RSA and Rabin. In Proc. Eurocrypt'96.
8. S. Bellovin and M. Merritt. Encrypted Key Exchange: Password-based Protocols Secure against Dictionary Attack. In Proc. IEEE Security and Privacy 1992.
9. D. Boneh, X. Ding, G. Tsudik, and C. Wong. A Method for Fast Revocation of Public Key Certificates and Security Capabilities. In Proc. Usenix Security Symposium 2001.
10. C. Boyd. Digital Multisignatures. Cryptography and Coding, H. J. Beker and F. C. Piper Eds., Clarendon Press, 1989, pp 241-246.
11. V. Boyko, P. MacKenzie, and S. Patel. Provably Secure Password Authentication and Key Exchange Using Diffie-Hellman. In Proc. Eurocrypt'00.
12. R. Cramer and V. Shoup. A Practical Public Key Cryptosystem Provably Secure Against Adaptive Chosen Ciphertext Attack. In Proc. Crypto'98.
13. D. Dean, T. Berson, M. Franklin, D. Smetters, and M. Spreitzer. Cryptography as a Network Service. In Proc. NDSS'01.
14. D. E. Denning. Digital Signature with RSA and other Public-Key Cryptosystems. C. ACM, 27(4), 1984, pp 388-392.
15. Y. Dodis, J. Katz, S. Xu, and M. Yung. Strong Key-Insulated Signature Schemes. In Proc. PKC'03.
16. A. Fiat and A. Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In Proc. Crypto'86.
17. R. Ganesan. Yaksha: Augmenting Kerberos with Public Key Cryptography. In Proc. NDSS'95.
18. O. Goldreich, S. Goldwasser, and S. Micali. How to Construct Random Functions. J. ACM, 33(4), 1986, pp 210-217.
19. S. Goldwasser, S. Micali, and R. Rivest. A Digital Signature Scheme Secure against Adaptive Chosen-Message Attacks. SIAM Journal on Computing, 17(2), 1988, pp 281-308.
20. G. Itkis and L. Reyzin. Forward-Secure Signatures with Optimal Signing and Verifying. In Proc. Crypto'01.
21. G. Itkis and L. Reyzin. SiBIR: Signer-Base Intrusion-Resilient Signatures. In Proc. Crypto'02.
22. J. Katz, R. Ostrovsky, and M. Yung. Efficient Password-Authenticated Key Exchange Using Human-Memorizable Passwords. In Proc. Eurocrypt'01.
23. P. MacKenzie and M. Reiter. Networked Cryptographic Devices Resilient to Capture. In Proc. IEEE Security and Privacy 2001.
24. P. MacKenzie, T. Shrimpton, and M. Jakobsson. Threshold Password-Authenticated Key Exchange. In Proc. Crypto'02.
25. T. Matsumoto, K. Kato, and H. Imai. Speeding Up Secret Computations with Insecure Auxiliary Devices. In Proc. Crypto'88.
26. R. Perlman and C. Kaufman. Secure Password-based Protocol for Downloading a Private Key. In Proc. NDSS'99.
27. B. Pinkas and T. Sander. Securing Passwords Against Dictionary Attacks. In Proc. ACM CCS'02.
28. C. Rackoff and D. Simon. Non-Interactive Zero-Knowledge Proof of Knowledge and Chosen Ciphertext Attack. In Proc. CRYPTO'91.
29. 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.
30. F. Schneider. Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial. ACM Comput. Surv. 22(4), 1990, pp 299-319.
31. S. Xu and R. Sandhu. Two Efficient and Provably Secure Schemes for Server-Assisted Threshold Signatures. In Proc. RSA Conference – Cryptographer's Track 2003.