Robust Software Tokens - Yet Another Method for Securing User's Digital Identity (*Extended and Revised : Ver. 0.9*)

Taekyoung Kwon

Sejong University, Seoul 143-747, Korea
tkwon@sejong.ac.kr

Abstract. This paper presents a robust software token that was developed to protect user's digital identity by simple software-only techniques. This work is closely related to Hoover and Kausik's software smart cards, and MacKenzie and Reiter's networked cryptographic devices, in the fact that user's private key is protected by postulating a remote server rather than tamper-resistance. The robust software token is aimed to be richer than the related schemes in terms of security, efficiency and flexibility. A two-party RSA scheme was carefully applied for the purpose, in a way of considering practical construction. This manuscript is an extended and revised version of the paper presented at ACISP 2003 conference.

1 Introduction

A public key infrastructure (PKI) appeared so promisingly, for example, for ensuring user's digital identity in a distributed environment. However, key management is still the issue both for public and private keys. Confidentiality should be assured for private keys while authenticity and validity are necessary for public keys [9]. A key roaming capability is another requisite for them [34, 17, 28]. In order for securing the private keys, a stable and portable container is required since those keys are not favorable to human memory. A *software token* which is stored in a user-controlled device, is widely accepted in general as a temporary alternative to the tamper-resistant device. However, such a password-encrypted token is susceptible to adversarial search in a small space of passwords [37]. Lately the networked methods were introduced, which could complicate the task of adversaries by postulating a remote server and its co-operation [26, 31], in spite that they respectively assumed different models and had some weakness [27].

The goal of this paper is to present a new *robust software token*, which is "robust" in the sense that no party can influence the output of the token cooperating with its remote server, beyond the influence exerted by one's own input. The robustness must be assured even if the token is captured in an hostile environment. We design the robust software token and its various protocols to be richer than the related schemes in terms of security, efficiency, and flexibility. This manuscript is an extended and revised version of the paper presented at ACISP 2003 conference [29]. Note that we have to correct the things related

to the assisted-open signature protocol since the protocol presented in [29] had weakness. So, in this extended and revised paper, the initialization process of the robust software token is more simplified, while the assisted-open signature protocol is revised as the practical version (and further extended to the provable version in [30]). In this paper, we add an appendix for discussing formal security of the provable version [30].

In Section 2 we will discuss the basic terms. Then we propose the method for generating the robust software token in Section 3. The robust software token protocols are proposed in Section 4 and Section 5, respectively. We analyze the proposed method in Section 6 and conclude this paper in Section 7. As we mentioned already, formal security of the provable version is discussed in appendix.

2 Preliminaries

In this section we summarize basic definition and terminology to be used in this paper. Subsequently we address security goals of our scheme.

2.1 Definitions

Parameters. We define two security parameters; a conventional parameter κ and a public key parameter ℓ, for example, $\kappa = 160$ and $\ell = 1024$. We could say the probability is negligible if it is less than $2^{-\kappa}$. Let us borrow some notation from [31]. So dvc and svr denote a user-controlled device and a trusted server, respectively. Also rst denotes the robust software token while vrf a verifier, π a password, pk_{svr} server's authentic public key, and sk_{svr} its corresponding private key. Finally θ will denote an X.509 certificate.

Adversaries. As defined in [31], let $\textsc{Adv}(T)$ denote the class of adversaries who compromised the elements of T, where $T \subseteq \{\mathsf{dvc}, \mathsf{svr}, \pi\}$, and $\textsc{Adv}(T_1) \subseteq \textsc{Adv}(T_2)$ if $T_1 \subseteq T_2$. The classes, $\textsc{Adv}(\{\mathsf{svr}\})$, $\textsc{Adv}(\{\mathsf{svr}, \pi\})$, $\textsc{Adv}(\{\mathsf{dvc},\mathsf{svr}\})$, $\textsc{Adv}(\{\mathsf{dvc}\})$, and $\textsc{Adv}(\{\mathsf{dvc}, \pi\})$ should be considered and analyzed if one can forge a signature.

RSA Signature. A digital signature scheme is $\Sigma = (\mathcal{G}_\Sigma(1^\ell), \mathcal{S}, \mathcal{V})$ where $\mathcal{G}_\Sigma$ is a probabilistic algorithm returning a public key pair from input 1^ℓ, and $\mathcal{S}$ and $\mathcal{V}$ are respectively signing and verifying algorithms, which run in polynomial time. We suppose to use a simple hash-and-sign RSA signature scheme in a probabilistic manner (with κ_Σ-bit random numbers), but any encoding scheme can be utilized before signing, for example, a standard encode-then-sign, a deterministic hash-and-sign or a Probabilistic Signature Scheme (PSS) can be used [3, 16]. The public key pair of a user is $\langle e, N\rangle$ and $\langle d, N, \phi(N)\rangle$ where N is the product of two large primes, $\phi(N)$ is the Euler totient function, and $ed \equiv 1 \bmod \phi(N)$ [38]. So, we assume $\mathcal{S}$ returns signature on a message m; $\langle s, r\rangle$ where $s \leftarrow H(m, r)^d \bmod N$ and $r \leftarrow_R \{0,1\}^{\kappa_\Sigma}$.

Encryption Schemes. We consider two encryption schemes in our system. One is a public key encryption scheme such as $\Gamma = (\mathcal{G}_\Gamma(1^\ell), \mathcal{E}, \mathcal{D})$, and the other is a conventional scheme such as $\Gamma_\Pi = (G, E, D)$. We assume $\mathcal{G}_\Gamma$, $\mathcal{E}$, G, and E are all probabilistic algorithms. We postulate $\mathcal{E}$ uses probabilistic encoding such as an Optimal Asymmetric Encryption (OAE) paddings [2, 7, 18, 40] or universally PSS [15], and E has random padding suitable to symmetric encryption.

Hash Functions. We consider two hash functions in our system. One is a function $f()$ for generating symmetric encryption keys from passwords, and the other is a function $H()$ for signature. $f()$ can be a pseudorandom function, for example, PBKDF2 of PKCS#5 [37]. For simplicity in our security argument, we assume $H()$ is a Full Domain Hash (FDH) function such as $H : \{0,1\}^* \rightarrow \mathbb{Z}_N^*$, meaning it hashes strings uniformly into the full domain $\mathbb{Z}_N^*$ [3]. The security of FDH is discussed in [3, 14].

2.2 Related Terms

Open PKI. An open PKI is a PKI where each certificate serves to establish authenticity in a potentially unbounded number of applications [9]. The generic notion of PKI implies the open PKI.

Closed PKI. A closed PKI is a PKI which has an influential issuer and clear contractual relationships among the issuer, certificate applicants, and verifiers [9]. In other words, it is the notion of private operation of the PKI framework. For example, a bank may run the closed PKI in a private manner that the bank is the issuer as well as the verifier, and its customers are the certificate applicants.

Two-Party RSA. Since Boyd introduced a multiparty RSA signature [8], a great amount of related work has been done [5, 6, 19, 21, 22, 36, 39]. The two-party RSA is the case that an RSA private key is split into two shares [5].

HK Model. Hoover and Kausik presented a pioneer study applying cryptographic camouflage to software tokens [26]. The idea was to make a private key unidentifiable by encrypting a random-looking part only. However, they had to encrypt user's public key and could not import the existing digital identity, for security reasons. Also impersonation was not considered carefully [27]. The HK model is a closed PKI model in which a client is a certificate applicant and a server is a verifier. So, two-factor authentication was most applicable [26].

MR Model. MacKenzie and Reiter presented the networked cryptographic devices along with a security proof [31]. They used the two-party RSA scheme in an additive manner, namely by splitting d into $d_1 + d_2$ of which one is a

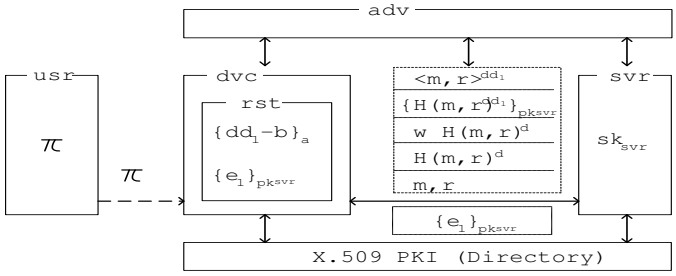

Fig. 1. Basic Concept

Ganensan's exponent [20], and allowed interesting features including key disable functions. However, adversarial replays or impersonation attempts could make a server busy without any detection [27]. It seems hard to obtain the untrusted server on the contrary to the original assumption [31]. The MR model is an open PKI model in which a server assists a client to perform a private key operation.

3 Robust Software Tokens

In this section we describe the basic idea and the method for generating rst. The robust software token is yet another method for securing user's digital identity in a user-controlled environment where the software token can be captured. The basic notion is given simply from two-party RSA derivation, $m^{dd_1} \rightarrow (m^{dd_1})^{e_1} \rightarrow m^d$ where mod N is omitted. Let us summarize our scheme briefly with Figure 1 before describing the details: (i) rst holds user's private key d, which was camouflaged[1] and encrypted under a password derived key a. (ii) dvc stores rst in its local memory and communicates with svr. (iii) A user controls dvc and relies on its output in hand. (S)he types the password π into dvc so as to run rst. (iv) svr does not maintain and cannot acquire any sensitive information about a user, but rather relies on PKIs. (v) An adversary has a dictionary of passwords and controls the whole network, meaning that (s)he can control the inputs to dvc and svr, and hear all of their outputs. However, (S)he cannot succeed in factoring N. Our scheme must consider the followings.

1. (Intrinsic Adversary) The adversary controls the whole network for the adversarial goals such as *chosen signature generation* or *private key recovery*. The empty class Adv({}) include those adversaries.
2. (Partial Compromise) The adversary who compromised either one of dvc and svr, should not gain more advantage than the intrinsic adversary. The classes, Adv({svr}) and Adv({dvc}), include those adversaries. Theoretically the

[1] Our camouflage method is different from that of Hoover and Kausik [26]. For example, our scheme does not need to encrypt user's public key $\langle e, N \rangle$, while their scheme needed it.

classes, $\textsc{Adv}(\{\mathsf{svr}, \pi\})$ and $\textsc{Adv}(\{\mathsf{dvc}, \pi\})$, should be observed in the same domain (see Appendix).

3. (Dictionary Attack) The adversary must compromise svr as well as dvc, i.e.,obtain rst and sk_{svr} together, in order to attempt the dictionary attack. The class $\textsc{Adv}(\{\mathsf{dvc},\mathsf{svr}\})$ includes those adversaries.
4. (Compatibility) Security must hold in the HK model and the MR model simultaneously. For example, a digital signature generated in the HK model should not degrade overall security in the MR model, and vice versa.

3.1 Robust Software Token Generation

The following rst-*generation* procedure can be performed off-line. So, even dvc as well as an authorized party could perform it without communicating directly with svr.

Stage 1. For initialization, a new RSA key pair must be generated or an existing key pair must be imported for a user. Then the values must be given, such as $\langle e, N\rangle$, $\langle d, N\rangle$, $\phi(N)$, π, and pk_{svr}. Note that $\phi(N)$ is necessary for algebraic conversion to two-party RSA, as it was in the related work [26, 31, 5]. Also the public key is presumed to be certified by a certificate authority (CA) [26, 31].

Stage 2. The following must be computed for generation of rst:

$$
\begin{aligned}
&e_1 \leftarrow_R \mathbb{Z}^*_{\phi(N)}\\
&d_1 \leftarrow e_1^{-1} \bmod \ \phi(N)\\
&a \leftarrow f(\pi)\\
&b \leftarrow_R \{0,1\}^1\\
&c \leftarrow (dd_1 \bmod \phi(N)) - b\\
&x \leftarrow E_a(c)\\
&y \leftarrow \mathcal{E}_{pk_{\mathsf{svr}}}(e_1)
\end{aligned}
$$

A new *friend exponent* e_1 is chosen at random, satisfying $1 \le e_1 < \phi(N)$ and $\gcd(e_1, \phi(N)) = 1$. Its modular inverse d_1 is called a *camouflage exponent*. Their lengths will be discussed in the following section. A symmetric cipher key a is derived from π by a key derivation function $f()$, for example, PBKDF of PKCS#5 [37]. Subsequently one bit integer b is chosen at random. Then d is multiplied by d_1 in $\mathbb{Z}^*_{\phi(N)}$, and b is subtracted from the product. The random-looking[2] value c is encrypted under a with random padding. The private key can be camouflaged in that sense. Finally the friend exponent e_1 is encrypted under pk_{svr} so that only svr can obtain e_1 afterward.

[2] Two random odd integers were multiplied in $\mathbb{Z}^*_{\phi(N)}$ and subtracted by one bit.

Stage 3. In general the CA may certify user's public key but svr can also do this if the closed PKI is only required in a way of restricting users to the closed domain. In that case y can be made more efficiently such that $y = E_\xi(e, e_1, N)$ even without a certificate where ξ is svr's symmetric cipher key. Finally all values such as $e, d, N, \phi(N), \pi, e_1, d_1, a, b, c, x, y,$ and pk_{svr}, must be erased from the generating device, while rst is given to the user. Note that rst is composed of the basic values, $\langle x, y, N\rangle$, the X.509 certificate θ for a user public key, and the extensive value pk_{svr}. Then rst is stored in dvc by software-only methods. Note that the pure private key, $\langle d, N, \phi(N)\rangle$, can optionally be stored in a tamper-resistant device, so as to be used for the same public key.

3.2 Discussion on Camouflage Parameters

In general it is desirable to choose a private exponent d being roughly the same size as N while we prefer to use a low public exponent e such as $2^{16} + 1$ [33, 41]. Let us discuss informally the length of camouflage parameters including e_1 and d_1. The friend exponent e_1 was encrypted so as to be read by svr only. It was treated as like a private exponent. So, the length of e_1 must be set to resist exhaustive search, since an adversary can derive M^d from M^{dd_1} if e_1 is so short as to find. Also e_1 must resist Wiener's attack for the case that dd_1 is compromised under a guessed password. Both for security and efficiency, we recommend 2κ bits for e_1, satisfying $e_1 > \sqrt[3]{N}$. Similarly it must be avoided that $d_1 < \sqrt[4]{N}$ for the opposite case. Fortunately it is unlikely that small d_1 will be generated from e_1 of length 2κ [33]. Also it could be obtained that dd_1 is greater than $\sqrt{N}$ due to the length of ee_1 such as $2\kappa + 17$.

Strictly speaking, the value c resides in $\mathbb{Z}_{\phi(N)}$ rather than $\mathbb{Z}^*_{\phi(N)}$ because b was one bit integer and subtracted from the $\phi(N)$-modular product, dd_1, of the values that are relatively prime to $\phi(N)$. Notice that $dd_1 (\mathrm{mod}\ \phi(N))$ must be an odd integer in $\mathbb{Z}^*_{\phi(N)}$. As a result, the least significant bit and the most significant bit of c, respectively, can be 0 or 1 equally likely. This is due to the modular multiplication dd_1 and the normal integer subtraction $-b$. The remaining bits are random as well. So, it might be secure to encrypt the value c with the password-derived key a in a way of cryptographic camouflage [26].

4 Robust Software Tokens in the HK Model

We can devise two kinds of protocols respectively for *two-factor authentication* and *closed digital signature* in the HK Model. Here we use the underlying RSA signature scheme in the slightly different way from $\mathcal{S}$ defined before. Also $H()$ is assumed as a normal hash function.

4.1 Two-factor Authentication

In the HK model, a limited number of servers should verify a closed signature using user's authentic public key. Though the original HK method encrypted

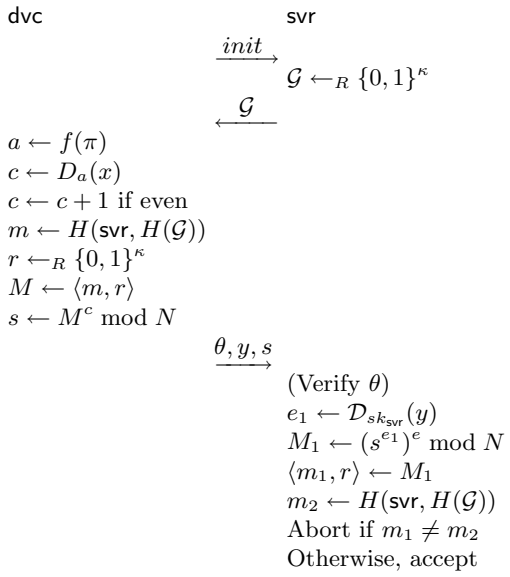

Fig. 2. Two-factor Authentication Protocol

the public key [26], our method can be released from such restriction. Figure 2 depicts the two-factor authentication protocol for using rst in the HK model.

Protocol Setup. A user knows π and has rst, i.e., storing rst in stable storage of dvc. Also svr may retain sk_{svr} in its storage device securely.

Protocol Run. An application may invoke rst, so that dvc sends an initiating signal to svr and the user types π on dvc. Upon receiving a random challenge $\mathcal{G}$ from svr, dvc derives a from π and recovers c by decrypting x. Here dvc must add 1 to c if it is even, meaning that dd_1 is recovered and denoted by c. Subsequently dvc computes $m = H(\mathsf{svr}, H(\mathcal{G}))$ and selects a random number r of κ bits. $H()$ denotes a strong one-way hash function. Then dvc concatenates m and r, and signs M with c, actually dd_1. Find this signature form, $\langle m, r\rangle^{dd_1}$, in Figure 1. Finally dvc sends $\langle \theta, y, s\rangle$ to svr. Then svr acquires the value e from θ in an authentic manner and the value e_1 by decrypting y. Note that θ is optional for the case that user's public key was certified by a CA. The value s is raised to e_1 and subsequently to e in $\mathbb{Z}_N$, meaning $(M^{dd_1})^{ee_1} \bmod N$. Finally svr obtains m_1 and verifies the signature with m_2 such that $m_2 = H(\mathsf{svr}, H(\mathcal{G}))$. If they match, svr authenticates the user.

Discussion. The reasons for signing $H(\mathsf{svr}, H(\mathcal{G}))$ are: (1) to prevent an adversary who masquerades svr, from getting a chosen message signed, and (2)

to keep s from being re-used illegally, for example, in the outside of svr's PKI domain. The random number r was included in the signature so as to keep an adversary from obtaining the chosen ciphertext s. That means, the adversary cannot acquire the same signature for a message even with a correct guess on c.

4.2 Closed Signature

It is trivial to derive an arbitrary signature generation protocol in the HK model. Simply dvc signs an arbitrary message m instead of $\mathcal{G}$ while only svr can verify it. Note that dvc must generate a signature in a form of $H(H(m), \mathsf{svr})$ for the same reasons above. Also signing with a timestamp may work against replays.

5 Robust Software Tokens in the MR model

In this section, we extend the MR model to devise three kinds of rst protocols for generating an RSA signature. The first will generate a signature simply by entrusting the signature completion to svr, so that it will be called an *entrusted open signature protocol*. The second will generate a signature with svr's assistance, as it did in the original MR scheme[3]. It means that svr assists the signature generation only and is not able to obtain a complete signature in the midst of the protocol run though the message could be understandable. So, we call it an *assisted open signature protocol*. The third is similar to the second except that svr is not able to understand a message when assisting the signature generation. In that sense, we call it a *blind-assisted open signature protocol*.

Common Requirements. (1) The open signature protocols must generate a signature that can be verified by user's authentic public key $\langle e, N \rangle$ in the open PKI. (2) Also svr must not be able to acquire any information about user's sensitive data such as a password or a private key in the protocol run. (3) Finally svr must be able to authenticate a user so as to discourage on-line attacks in the protocol run. Specific requirements will be added in the following sections. The main difference stems from the degree of user's trusts on svr[4] in each protocol.

Common Protocol Setup. In addition to the previous protocol setup, a verifier vrf is able to access and verify user's certificate θ.

[3] It was a misconception to assume the untrusted server in the original MR scheme where the server was not able to detect impersonation or replay of an adversary [31, 27]. Our assisted signature schemes could allow the untrusted server under postulating a PKI support (See Figure 4 and 5). In spite of the improvement, we recommend the server be trusted because it is unlikely to assume that the untrusted server would not deny the service.

[4] If svr is compromised, the service denial is unavoidable. This is the shared weakness of all the related work that exploits network connectivity [26, 31].

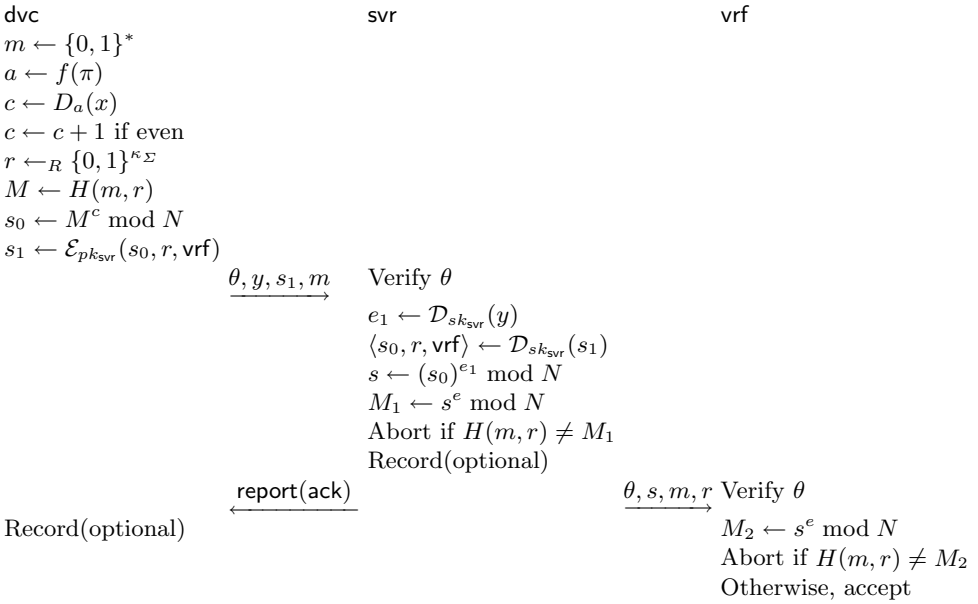

Fig. 3. Entrusted Open Signature Protocol

5.1 Entrusted Open Signature

This protocol is the most simple and efficient one for using rst in the MR model. See Figure 3. The basic notion is that svr is trusted, so as to complete user's open signature and transmit the signature to vrf on behalf. When we set vrf = dvc in s_1, svr may return the complete signature to dvc. Also vrf can be set as a list of many verifiers. Possible application may include the signed e-mail delivery.

Protocol Run. Given the inputs such as π, vrf and m where m means a message, dvc should derive a from π and decrypt x to obtain the value c. Also dvc should add one if it is even. The random number r is chosen at length κ_Σ. The hash value $H(m,r)$ is derived and signed with c. Then $\langle s_0, r, \mathsf{vrf}\rangle$ is encrypted with pk_{svr}. Subsequently dvc sends $\langle \theta, y, s_1, m\rangle$ to svr. Then svr decrypts y and s_1 so as to acquire e_1 and s_0, respectively. Here svr can generate a complete signature s by using e_1 and s_0, and verify it with user's public key $\langle e, N\rangle$. If s is a valid signature, svr on behalf of dvc, transmits it to vrf and reports the result. Note that svr and dvc can log the record to meet the requirement of non-repudiation. Finally vrf verifies user's signature with θ.

Discussion. The reason for encrypting s_0 and r under pk_{svr} is to keep an adversary, acquired rst and s_1, from running dictionary attacks. A time-stamp t is necessary only if retransmission of s needs to be prevented, because replaying s_1

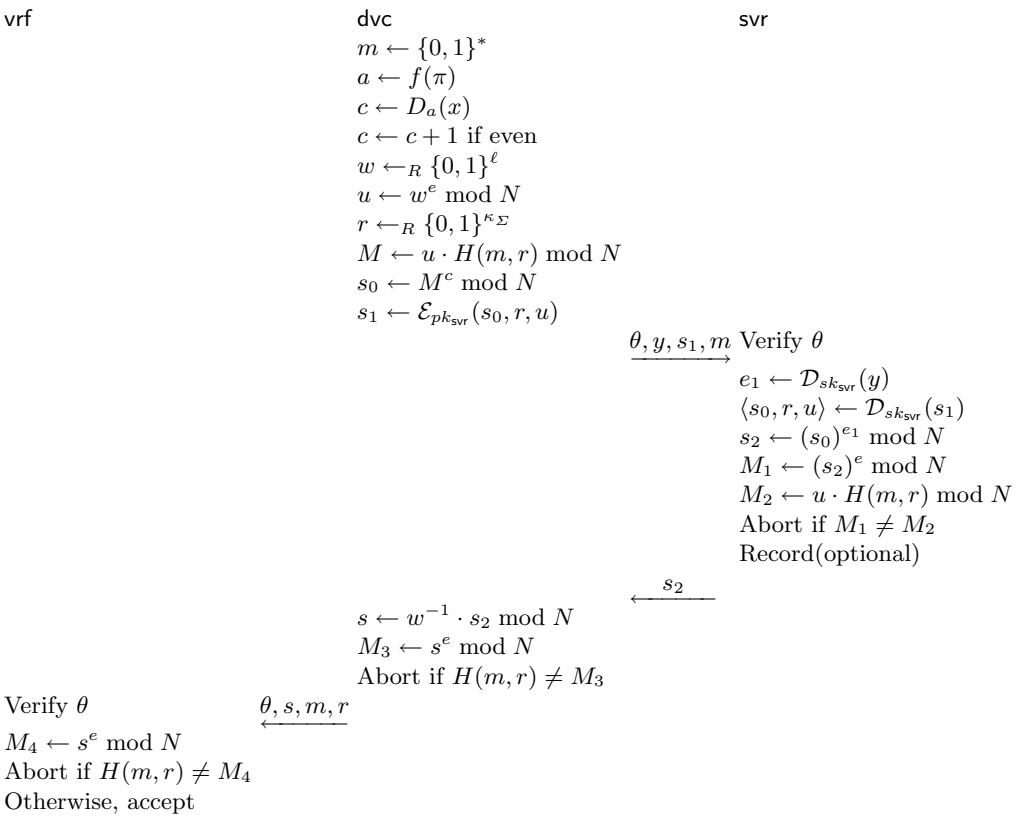

Fig. 4. Assisted Open Signature Protocol

cannot change s. In practice, we can embed a time-stamp into the random value r (for example, r is concatenated with time-stamp t where $r \leftarrow_R \{0,1\}^{\kappa_\Sigma - |t|}$).

5.2 Assisted Open Signature

Figure 4 depicts the assisted open signature protocol for rst. In addition to the common requirements, this protocol must meet the followings requirements as well: (1) dvc must complete the signature generation, and (2) svr should not acquire a complete signature in the midst of the protocol run. The protocol setup is enhanced that svr is not allowed to complete the signature generation.

Protocol Run. Again dvc derives a from π and decrypts x to obtain the odd integer c. The random number r is chosen at length κ_Σ. The hash value $H(m,r)$ is derived from a message m while the random number w is chosen from $\mathbb{Z}_N^*$. Then w is raised to e and we call this u. The value u is multiplied by $H(m,r)$ in

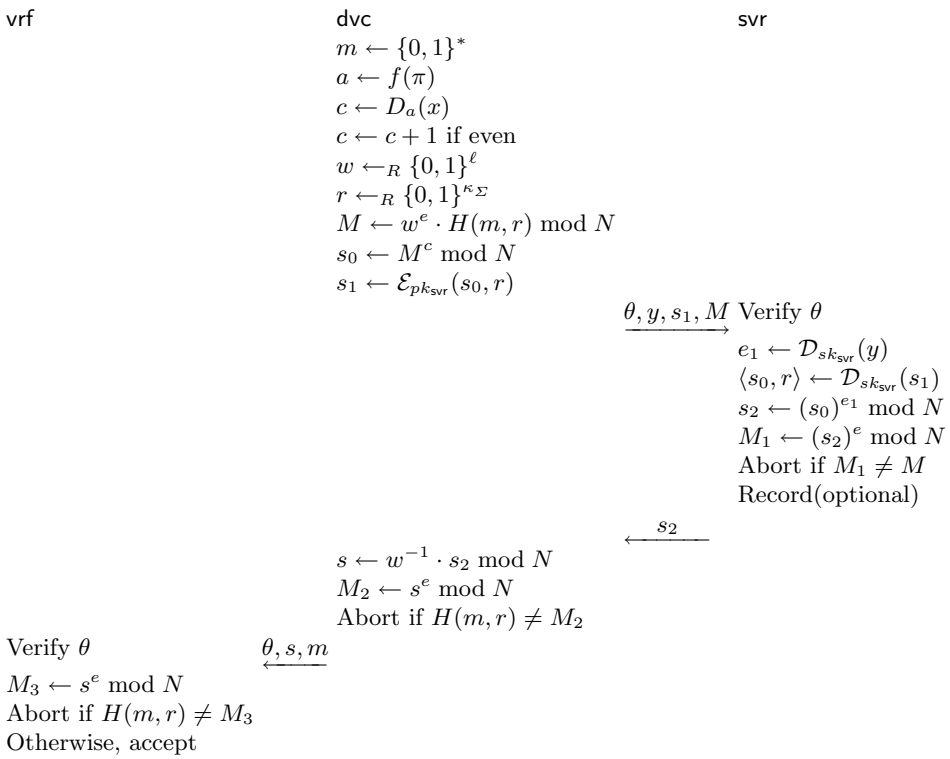

Fig. 5. Blind-Assisted Open Signature Protocol

$\mathbb{Z}_N$. Then the product M is signed with c. Here $\langle s_0, r, u\rangle$ is encrypted under pk_{svr}. Subsequently dvc sends $\langle \theta, y, s_1, m\rangle$ to svr. Then svr decrypts y and s_1 to obtain e_1, u and s_0. The value M_1 is obtained by raising s_0 to e_1 and then e in $\mathbb{Z}_N$, while the value M_2 by multiplying u and $H(m,r)$ in $\mathbb{Z}_N$. If M_1 and M_2 match, svr sends s_2 to dvc. Also svr can log the record for satisfying non-repudiation. Upon receiving s_2, dvc multiplies it by w^{-1} in $\mathbb{Z}_N$ and verifies the result using $\langle e, N\rangle$. If s is a valid signature, dvc could transmit it to vrf which will verify it.

Discussion. All discussion held on the entrusted signature also applies in this case. In addition, the reason for signing $u \cdot H(m,r)$ is to keep svr from completing the signature generation or obtaining s before dvc finishes the protocol run. Note that s is not derivable from s_2 without v or d.

5.3 Blind-Assisted Open Signature

A blind signature scheme, first introduced by Chaum, allows a user to get a message signed by another party without revealing any information about the

message [12]. We can apply this technique to the assisted open signature protocol, so as to keep svr from understanding the message. Figure 5 depicts the blind-assisted open signature protocol in that sense. An additional requirement is that svr should not acquire any information about a message when assisting the signature generation. So, the protocol setup is enhanced again that svr is not allowed to read the original message.

Protocol Run. Similarly dvc derives a from π and decrypts x to obtain the odd integer c. The hash value $H(m, r)$ is derived from a message m while the random number w is chosen from $\mathbb{Z}_N^*$. Then w is raised to e and multiplied by $H(m, r)$ in $\mathbb{Z}_N$. The product M is signed with c. Note that we can replace $H(m, r)$ with m in all open signature protocols. The random number r is chosen at length κ. Here $\langle s_0, r\rangle$ is encrypted under pk_{svr}. Subsequently dvc sends $\langle \mathcal{C}, y, s_1, M\rangle$ to svr. Then svr decrypts y and s_1 so as to obtain e_1 and s_0, respectively. The value M_1 is acquired by raising s_0 to e_1 and then e in $\mathbb{Z}_N$. If M and M_1 match, svr sends s_2 to dvc. Also svr can log the record for satisfying non-repudiation. Upon receiving s_2, dvc multiplies it by w^{-1} in $\mathbb{Z}_N$ and verifies the result using user's public key, $\langle e, N\rangle$. If s is a valid signature, dvc transmits it to vrf. Finally vrf can verify user's signature by obtaining $\langle e, N\rangle$ from θ.

Discussion. All discussion held on the entrusted signature applies in this case as well. In addition, the reason for signing $w^e \cdot H(m, r)$ is to keep svr from understanding $H(m, r)$ in a notion of the blind signature. This protocol can derive a RSA decryption protocol using rst in that sense.

6 Analysis and Comparison

6.1 Security

Security of the proposed scheme was informally discussed in Section 3.2 and each discussion part in Section 4 and 5. Notice that rst retained user's components in the form of $\langle E_a(dd_1 - b), N, \mathcal{E}_{pk_{\mathsf{svr}}}(e_1)\rangle$, satisfying the notion of cryptographic camouflage. The most significant bit and the least significant bit of "$dd_1 (\mathrm{mod}\ \phi(N)) - b$" are not predictable, while the remaining bits are random as well. The friend exponent e_1 is encrypted under pk_{svr} and handled carefully with d_1. As for each protocol run, we were able to see that the flexible goals were respectively attained. Let us discuss formal security of the proposed scheme in Appendix A.

6.2 Efficiency

If the RSA exponent is chosen at random, RSA encryption using the repeated square-and-multiply algorithm will take ℓ modular squarings and expected $\frac{\ell}{2}$ modular multiplications [33]. So, we can evaluate the computational performance of the networked software token methods, with regard to the number of modular

Number of Multiplications	*Client* (dvc)	*Server* (svr)
HK Scheme-1 [26]	1.5ℓ	$2.25\ell + (\iota + 1)$
HK Scheme -2 [26]	$1.5(\iota + 1)\ell$	$1.5\ell + (1.5\kappa + \iota + 1)$
RST (Two-factor Auth.)	1.5ℓ	$1.5\ell + (3\kappa + 2\iota + 2)$
MR Scheme [31]	$1.5\ell + (3\iota + 4)$	9ℓ
Augmented MR Scheme [27]	$1.5\ell + (3\iota + 4)$	$10.5\ell + \iota + 1$
RST (Entrusted)	$1.5\ell + (2\iota + 2)$	$4.5\ell + (3\kappa + 2\iota + 2)$
RST (Assisted)	$1.5\ell + (5\iota + 7)$	$6\ell + (3\kappa + 2\iota + 3)$
RST (Blind-Assisted)	$1.5\ell + (4\iota + 6)$	$4.5\ell + (3\kappa + 2\iota + 2)$

Table 1. Performance Evaluation of Software Token Methods (m excluded)

N multiplications. See Table 1. Define a new parameter $\iota = 16$, and assume $e = pk_{\mathsf{svr}} = 2^{16} + 1$. Without losing impartiality, the costs for certificate verification $(\iota + 1)$ were counted in HK schemes and all of our schemes. Also the length of message m was not counted for encryption in the MR and its augmented scheme, since our scheme posed it outside the encryption. In that sense, we do not lose impartiality in the comparison. All RSA moduli are assumed ℓ bits for simplicity. Note that the underlying public key encryption scheme is assumed RSA and its encoding is OAEP or OAEP+, i.e., the maximum length n of a message in one modular N block is less than ℓ such as $n < \ell - k_0 - k_1$[2]. Also we have implemented three assisted signature protocols and found the similar result to Table 1 [29].

7 Conclusion

This manuscript is an extended and revised version of the paper presented at ACISP 2003 conference. In this paper, we introduced a simple method called the robust software token in order to protect user's digital identity by software-only techniques. Authentication and various types of signature generation are possible applications for this scheme under postulating a PKI support to user-controlled environments. A networked device, which holds the robust software token, could be more promising for ubiquitous computing environment. We considered security, efficiency, and flexibility while designing the software token in the way to improve various features of the invaluable predecessors such as the software smart cards [26] and the networked cryptographic devices [31]. The robust software token can be used flexibly in the MR model as well as the HK model, without ignoring security and efficiency. It is so *robust* that only the legitimate party can generate or update it even off-line while it must be used on-line afterwards. A discretionary control of the legitimate party can be made to the degree of server assistance. In order for describing the schemes, practical construction was more considered than theoretical framework in this paper. Formal security of the proposed scheme is handled in the separate paper [30] and summarized in Appendix A of this paper.

Acknowledgement

I would like to thank anonymous referees for their helpful comments.

References

1. M. Bellare and P. Rogaway, "Random oracles are practical: A paradigm for designing efficient protocols," In Proceedings of *ACM Conference on Computer and Communications Security*, pp.62-73, Nov. 1993.
2. M. Bellare and P. Rogaway, "Optimal asymmetric encryption," *Advances in Cryptology - Eurocrypt*'94, Lecture Notes in Computer Science, Vol. 950, Springer-Verlag, pp.92-111, 1995.
3. M. Bellare and P. Rogaway, "The exact security of digital signatures - How to sign with RSA and Rabin," *Advances in Cryptology - Eurocrypt*'96, Lecture Notes in Computer Science, Vol. 1070, Springer-Verlag, pp.399-416, 1996.
4. M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway, "Relations among notions of security for public-key encryption schemes," *Advances in Cryptology - Crypto*'98, Lecture Notes in Computer Science, Vol. 1462, Springer-Verlag, pp.26-45, 1998.
5. M. Bellare and R. Sandhu, "The security of practical two-party RSA signature schemes," Manuscript, 2001.
6. D. Boneh and M. Franklin, "Efficient generation of shared RSA keys," *Advances in Cryptology - Crypto*'97, Lecture Notes in Computer Science, Vol. 1294, Springer-Verlag, pp.425-439, 1997.
7. D. Boneh, "Simplified OAEP for the RSA and Rabin functions," *Advances in Cryptology - Crypto* 2001, Lecture Notes in Computer Science, Vol. 2139, Springer-Verlag, pp.275-291, 2001.
8. C. Boyd, "Digital multisignatures," *Cryptography and Coding*, Oxford University Press, pp.241-246, 1989.
9. S. Brands, *Rethinking public key infrastructures and digital certificates*, The MIT Press, p.11 and pp.219-224, 2000.
10. E. Brier, C. Clavier, J. Coron, and D. Naccache, "Cryptanalysis of RSA signatures with fixed-pattern padding," *Advances in Cryptology - Crypto* 2001, Lecture Notes in Computer Science, Vol. 2139, pp.433-439, 2001.
11. R. Canetti, O. Goldreich, and S. Halevi, "The random oracle methodology, revisited," in Proc. 30th Annual ACM Symposium on Theory of Computing (STOC), pp. 209–218, 1998.
12. D. Chaum, "Blind signatures for untraceable payments," *Advances in Cryptology - Crypto*'82, Lecture Notes in Computer Science, Vol. 1440, Springer-Verlag, pp.199-203, 1983.
13. D. Coppersmith, "Finding a small root of a univariate modular equation," *Advances in Cryptology - Eurocrypt*'96, Lecture Notes in Computer Science, Vol. 1070, Springer-Verlag, pp.155-165, 1996.
14. J. Coron, "On the exact xecurity of full domain hash," *Advances in Cryptology - Crypto* 2000, Lecture Notes in Computer Science, Vol. 1880, pp.229-235, 2000.
15. J. Coron, M. Joye, D. Naccache, and P. Paillier, "Universal padding schemes for RSA," *Advances in Cryptology - Crypto* 2002, Lecture Notes in Computer Science, Vol. 2442, pp.226-241, 2002.

16. D. Denning, "Digital signatures with RSA and other public key cryptosystems," *Communications of the ACM*, Vol.27, No.4, pp.388-392, Apr. 1984.
17. W. Ford and B. Kaliski, "Server-assisted generation of a strong secret from a password," In Proceedings of the *International Workshops on the Enabling Technologies: Infrastructure for Collaborative Enterprise*, IEEE, June 2000
18. E. Fujisaki, T. Okamoto, D. Pointcheval, and J. Stern, "RSA-OAEP is secure under the RSA assumption," *Advances in Cryptology - Crypto* 2001, Lecture Notes in Computer Science, Vol. 2139, Springer-Verlag, pp.260-274, 2001.
19. Y. Frankel, P. MacKenzie, and M. Yung, "Robust efficient distributed RSA key generation," In Proceedings of the *ACM Symposium on Theory of Computing*, pp.663-672, 1998.
20. R. Ganesan, "Yaksha: Augmenting Kerberos with public key cryptography," In Proceedings of the *ISOC Network and Distributed System Security Symposium*, February 1995.
21. R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Robust and efficient sharing of RSA functions," *Advances in Cryptology - Crypto*'96, Lecture Notes in Computer Science, Vol. 1109, Springer-Verlag, pp.157-172, 1996.
22. N. Gilboa, "Two party RSA key generation," *Advances in Cryptology - Crypto*'99, Lecture Notes in Computer Science, Vol. 1666, Springer-Verlag, pp.116-129, 1999.
23. O. Goldreich, S. Goldwasser, and S. Micali, "How to construct random functions," *Journal of the ACM*, vol.33, no.4, pp.792-807, Oct. 1984.
24. O. Goldreich, *Foundations of Cryptography - Basic Tools*, Cambridge University Press, 2001.
25. S. Goldwasser, S. Micali, and R. Rivest, "A digital signature scheme secure against adaptive chosen-message attacks," *SIAM Journal of Computing*, vol.17, no.2, pp.281-308, Apr. 1988.
26. D. Hoover, B. Kausik, "Software smart cards via cryptographic camouflage," In Proceedings of the *IEEE Symposium on Security and Privacy*, 1999.
27. T. Kwon, "On the difficulty of protecting private keys in software," *Information Security - ISC 2002*, Lecture Notes in Computer Science, Vol. 2433, Springer-Verlag, September 2002. A revised version can be found at `http://dasan.sejong.ac.kr/~tkwon/research/difficulty.pdf`.
28. T. Kwon, "Virtual Software Tokens - A practical way to secure PKI roaming," *Infrastructure Security - InfraSec 2002*, Lecture Notes in Computer Science, Vol. 2437, Springer-Verlag, October 2002.
29. T. Kwon, "Robust Software Tokens - Yet another method for securing user's digital identity," *Information Security and Privacy - ACISP 2003*, Lecture Notes in Computer Science, Vol. ????, Springer-Verlag, July 2003.
30. T. Kwon, "On securing RSA signature keys in networked environments," *Currently Unpublished Manuscript*, 2003.
31. P. MacKenzie and M. Reiter, "Networked cryptographic devices resilient to capture," In Proceedings of the *IEEE Symposium on Security and Privacy*, 2001, a full and updated version is DIMACS Technical Report 2001-19, May 2001.
32. P. MacKenzie and M. Reiter, "Delegation of cryptographic servers for capture-resilient devices," In Proceedings of the *ACM Conf. on Computer and Communications Security*, Nov. 2001, a full and updated version is DIMACS Technical Report 2001-37, Nov. 2001.
33. A. Menezes, P. van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*, CRC Press, pp.287-291, pp.312-315, 1997.

34. R. Perlman and C. Kaufman, "Secure password-based protocol for downloading a private key," In Proceedings of the *ISOC Network and Distributed System Security Symposium*, February 1999.
35. D. Pointcheval and J. Stern, "Security proofs for signature schemes," *Advances in Cryptology - Eurocrypt*'96, Lecture Notes in Computer Science, Vol. 1070, Springer-Verlag, pp.387-398, 1996.
36. G. Poupard and J. Stern, "Generation of shared RSA keys by two parties," *Advances in Cryptology - Asiacrypt*'98, Lecture Notes in Computer Science, Vol. 1514, Springer-Verlag, pp.11-24, 1998.
37. PKCS #5, "Password-based encryption standard," *RSA Laboratories Technical Note*, Version 2.0, 1999.
38. R. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol.21, pp.120-126, 1978.
39. V. Shoup, "Practical Threshold Signatures," *Advances in Cryptology - Eurocrypt*'00, Lecture Notes in Computer Science, Vol. 1807, Springer-Verlag, pp.207-220, 2000.
40. V. Shoup, "OAEP reconsidered," *Advances in Cryptology - Eurocrypt* 2001, Lecture Notes in Computer Science, Vol. 2139, Springer-Verlag, pp.239-259, 2001.
41. M. Wiener, "Cryptanalysis of short RSA secret exponents," *IEEE Transactions on Information Theory*, vol.36, no.3, May 1990.

A Security Analysis

In this section we provide a formal security proof for the robust software token methods in the random oracle model where hash functions are assumed to be ideal random functions. The proof idea is based on the MR security model where a simulation argument is well achieved [31], so that we handle the related scheme, i.e., the assisted open signature protocol. The robust software token must be "robust" in the sense that no party can influence the output of the token cooperating with its remote server, beyond the influence exerted by one's own input. So, we prove that if a forger breaks the robust software token system (later we will redefine the provable version as T-RSA) with probability non-negligibly more than what is inherently possible, then the underlying signature scheme (under a chosen message attack of an existential forger[5] [25, 35]) or encryption scheme (under an adaptive chosen ciphertext attack[6], IND-CCA2 [4]) can be

[5] A forger, $\mathcal{F}$, is given pk and allowed to query a corresponding signature oracle on messages of its choice, in order to forge signatures with respect to pk, where $(pk, sk) \leftarrow \mathcal{G}_{\Sigma}(1^{\ell})$. We say a forger, $\mathcal{F}$, (q, ϵ)-breaks Σ if it makes q queries to the signature oracle, and outputs a valid forgery (m, σ) with probability at least ϵ, where m was not one of the messages previously signed by the signature oracle.

[6] An attacker, A, is given pk' and allowed to query a corresponding decryption oracle on ciphertexts, where $(pk', sk') \leftarrow \mathcal{G}_{\Gamma}(1^{\ell})$. At some point, A generates two equal length strings X_0 and X_1, and submits them to the test oracle, which will draw $b \leftarrow_R \{0,1\}^1$, and return $Y = \mathcal{E}_{pk'}(X_b)$. Then A continues as before but without querying Y. Finally A outputs b' and wins if $b' = b$. We say an attacker, A, (q, ϵ)-breaks Γ if it makes q queries to the decryption oracle, and $2 \cdot \Pr[b' = b] - 1 \geq \epsilon$.

broken with non-negligible probability, by constructing such a forger or attacker. The proof is made on the various types of adversaries.

A.1 Provable Version [30]

The provable version of the robust software token methods is described in [30]. It is a slightly modified version of the assisted-open signature protocol to be proved in the MR security model[31]. We call it T-RSA (Two-party RSA) by which a device generates an RSA signature on a message m with assistance of a server as follows:

$T_{\langle d,N\rangle}(m)$: $r \leftarrow_R \{0,1\}^{\kappa_\Sigma}$; $s \leftarrow H(m,r)^d \bmod N$; return $\langle s, r\rangle$

In the T-RSA system, the robust software token is initialized (we added the key disabling capability of the MR scheme to the T-RSA system), and the modified assisted-open signature protocol is executed between the device and the server in order to generate an RSA signature.

Device Initialization The following procedure can be performed off-line for device initialization, i.e., the initialization of the robust software token. So, an authorized party or the device could perform it for a corresponding user, without communicating directly with the server.

A new RSA key pair must be generated or an existing key pair must be imported for a user. The key pair means $\langle e, N\rangle$ and $\langle d, N, \phi(N)\rangle$. Also the user's password π and the server's public encryption key pk_{svr} must be provided. The public keys are presumed to be certified by a certificate authority (CA), for example, an X.509 certificate θ for $\langle e, N\rangle$.

The following must be computed for initialization:

$$
\begin{aligned}
&e_1 \leftarrow_R \mathbb{Z}^*_{\phi(N)}\\
&d_1 \leftarrow e_1^{-1} \bmod \ \phi(N)\\
&a \leftarrow f(\pi)\\
&b \leftarrow_R \{0,1\}^1\\
&c \leftarrow (dd_1 \bmod \phi(N)) - b\\
&x \leftarrow E_a(c)\\
&y \leftarrow \mathcal{E}_{pk_{\mathsf{svr}}}(e_1)\\
&t \leftarrow_R \{0,1\}^\kappa\\
&v \leftarrow H_{\mathsf{dsbl}}(e_1, t)\\
&z \leftarrow \mathcal{E}_{pk_{\mathsf{svr}}}(t, v)
\end{aligned}
$$

A new exponent e_1 is drawn randomly from $\mathbb{Z}^*_{\phi(N)}$ so that $1 \le e_1 < \phi(N)$ and $\gcd(e_1, \phi(N)) = 1$. Its modular inverse d_1 is set as a *camouflage exponent*. A symmetric cipher key a is derived from π by a key derivation function $f()$. Subsequently one bit integer b is chosen at random. Then d is multiplied by d_1 modulo $\phi(N)$, and b is subtracted from it. The random-looking value c is encrypted under a and called x in the sense of camouflaging. Eventually the

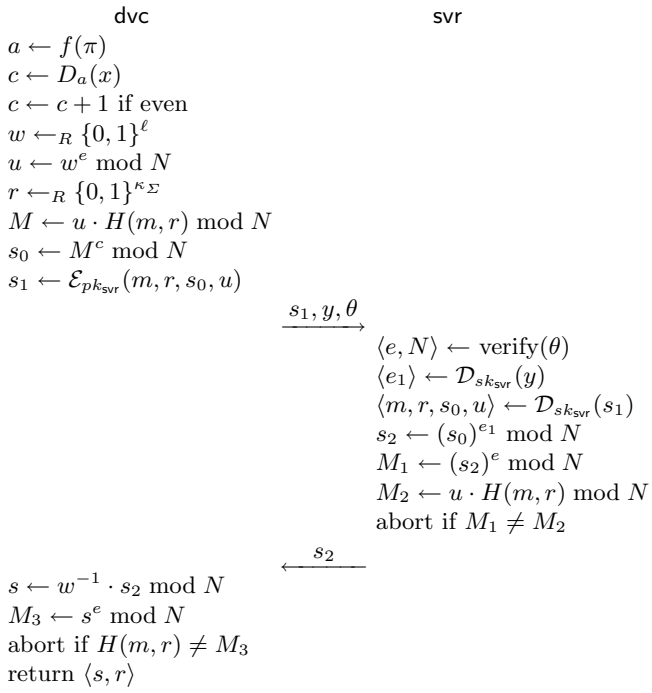

Fig. 6. Provable Version (T-RSA)

exponent e_1 is encrypted under pk_{svr} and called y, so that only svr can obtain e_1 afterward. For a key disabling capability (the nice idea of MR [31]), t is chosen at random and hashed with e_1 to v. The values t and v are encrypted under pk_{svr} and called z.

Finally the values x and y are saved on stable storage in the device while y and z are backed up off-line for the case that key disabling is necessary. All sensitive values such as d, $\phi(N)$, π, e_1, d_1, a, b, c, t, and z are deleted from the device. Also pk_{svr} and θ of $\langle e, N\rangle$ may be stored in the device. Note that the private key, $\langle d, N, \phi(N)\rangle$, can be stored in a tamper-resistant device to be used for the same public key.

Signature Protocol Figure 1 depicts the T-RSA protocol. The input to dvc is the password π and the message m such that $m \leftarrow \{0,1\}^*$. Then dvc derives a from π and decrypts x to obtain the odd integer c (add 1 if it was even). The random value r is a κ_Σ-bit integer chosen for a probabilistic signature. w is drawn randomly from $\mathbb{Z}_N$ and raised to e for being called u. The value u is multiplied by the hash value $H(m,r)$ and the result M is raised to c in $\mathbb{Z}_N$. s_1 is an encryption of m, r, s_0 and u under pk_{svr} for secure transmission. Subsequently dvc sends s_1, y, and an X.509 certificate θ to svr. Then svr verifies θ and decrypts

y and s_1 to obtain e_1, m, r, s_0 and u. M_1 is obtained by raising s_0 to e_1 and e in $\mathbb{Z}_N$. M_2 is by multiplying u and $H(m, r)$ modulo N. If M_1 and M_2 are equal, svr sends s_2 to dvc. Upon receiving s_2, dvc multiplies it by w^{-1} modulo N and verifies the result using $\langle e, N\rangle$. If s is a valid signature, dvc returns $\langle s, r\rangle$.

If the user wants to disable the private key, for example, since the device has been stolen, (s)he should send y and z to svr. Then svr could obtain e_1, t, and v from them, and update its *disabled list* if $v = H_{\mathsf{dsbl}}(e_1, t)$.

A.2 Assumptions

A forger is given $\langle e, N\rangle$ and possibly θ. In order to prove security formally, we define oracles such as a dvc oracle, a srv oracle, and random oracles f and H. Without loss of generality, we assume for the random oracles that a new query is answered with a fresh random output (a random hash of its input) in the defined range, but an old query is answered consistently with the previous queries. In our scheme, a symmetric key is used for encrypting c, and is derived by f. For convenience in our proof, we could assume f as a random oracle such that $f : \{0,1\}^* \to \mathbb{Z}_N^*$ and once its output was c for input π, for a specific case that x and c can be verified by the querier. As defined in [31], the dvc oracle may be queried with start and finish. On start(m), it returns $\langle s_1, y, \theta\rangle$, and sets some internal state. On the corresponding finish(s_2), it either aborts or returns a valid signature for m. The srv oracle may be queried with serve(s_1, y, θ), and either aborts or returns s_2. Let $\bar{q} = (q_{\mathsf{dvc}}, q_{\mathsf{srv}}, q_o, q_f, q_H)$ where q_{dvc} is the number of start queries, q_{srv} that of serve, q_f and q_H those of random oracles, and q_o that of others not counted above. Though we omit the key disabling capability in our proof, it is trivial to add it to our description.

We say that a forger wins if after those queries it can output a valid signature pair $(m, \langle s, r\rangle)$ and there was no start(m) query. Note that a type Adv({dvc, π}) forger only wins if there was no serve(s_1, y, θ) query for the valid output. We say a forger $(\bar{q}, \epsilon)$-breaks T-RSA if it makes the number of $\bar{q}$ oracle queries and wins with probability at least ϵ.

A.3 Security of T-RSA

We prove theorems for the classes, Adv({svr}), Adv({svr, π}), Adv({dvc,svr}), Adv({dvc}), and Adv({dvc, π}), if one in those classes is an existential forger which forges RSA signature from the T-RSA scheme. Our security argument is made on the well-constructed framework of MR [31], so that the similar theorems can be proved in the random oracle model.

In Theorem 1, we are given $\mathcal{F} \in$ Adv({svr}) that $(\bar{q}, \epsilon)$-breaks the T-RSA system, and then we will construct a forger $\mathcal{F}^*$ for the underlying RSA signature scheme. We assume that θ is given for all.

Theorem 1. *If a type* Adv({svr}) *forger* $(\bar{q}, \epsilon)$*-breaks the T-RSA system, then there exists a forger that* $(q_{\mathsf{dvc}}, \epsilon')$*-breaks the underlying RSA signature scheme with* $\epsilon' \approx \epsilon$.

Proof. $\mathcal{F}^*$ is given public key $\langle e, N\rangle$ for the RSA signature scheme and simulates the T-RSA system for a forger $\mathcal{F}$, so that any forgery constructed by $\mathcal{F}$ will be a forgery in the underlying RSA signature scheme.

$\mathcal{F}^*$ generates the server's key pair $(pk_{\mathsf{svr}}, sk_{\mathsf{svr}})$, the user password $\pi \leftarrow_R \Pi$, and finally x and y in the normal way, except that e_1, d_1 and c are drawn randomly from $\mathbb{Z}_N$. $\mathcal{F}^*$ gives $\langle e, N\rangle$ and $(pk_{\mathsf{svr}}, sk_{\mathsf{svr}})$ to $\mathcal{F}$.

$\mathcal{F}^*$ responds to a serve query as in the real protocol. $\mathcal{F}^*$ responds to f and H queries as a normal random oracle would.

$\mathcal{F}^*$ responds to a $\mathsf{start}(m)$ query by querying the signature oracle to get $\sigma = \langle s, r\rangle$ where $s = H(m, r)^d \bmod N$ and then responding as in the real protocol, except that it sets w and s_0 as follows:

$\rho \leftarrow_R \mathbb{Z}_N$
$w \leftarrow \rho^{ce_1} \cdot s^{(e_1 d_1 - 1)}$
$s_0 \leftarrow \rho^c \cdot s^{d_1} \bmod N$

Note that M is still $u \cdot H(m, r) \bmod N$ where $u = w^e$ but $s_0 \neq M^c \bmod N$.

$\mathcal{F}^*$ responds to a $\mathsf{finish}(s_2)$ query corresponding to a $\mathsf{start}(m)$ query as in the real protocol.

Analysis: From $\mathcal{F}$'s perspective, the above simulation is statistically indistinguishable from the real protocol due to the randomness of the referred values. So, if $\mathcal{F}$ produced a forgery with probability ϵ in T-RSA, then it can forge with probability $\epsilon' \approx \epsilon$ in this simulation. To break the underlying RSA signature scheme with probability at least $\epsilon' \approx \epsilon$, $\mathcal{F}^*$ may run $\mathcal{F}$ in this simulation and output any forgery produced by $\mathcal{F}$. □

The strategy on making the simulation indistinguishable to $\mathcal{F}$ was to make $\mathcal{F}^*$ set w and s_0 artificially in the defined distribution. This result is exactly the same even if π is additionally obtained by the adversary, because π does not affect any distribution in the above simulation.

Theorem 2. *If a type* $\textsc{Adv}(\{\mathsf{svr}, \pi\})$ *forger* $(\bar{q}, \epsilon)$*-breaks the T-RSA system, then there exists a forger that* $(q_{\mathsf{dvc}}, \epsilon')$*-breaks the underlying RSA signature scheme with* $\epsilon' \approx \epsilon$.

Proof. The proof is manifest due to Theorem 1. □

In Theorem 3, we are given $\mathcal{F} \in \textsc{Adv}(\{\mathsf{dvc}, \mathsf{svr}\})$ that $(\bar{q}, \epsilon)$-breaks the T-RSA system, and then we will construct a forger $\mathcal{F}^*$ for the underlying RSA signature scheme.

Theorem 3. *If a type* $\textsc{Adv}(\{\mathsf{dvc}, \mathsf{svr}\})$ *forger* $(\bar{q}, \epsilon)$*-breaks the T-RSA system, then there exists a forger that* $(q_{\mathsf{dvc}}, \epsilon')$*-breaks the underlying RSA signature scheme with* $\epsilon' \approx \epsilon - \frac{q_f}{|\Pi|}$.

Proof. $\mathcal{F}^*$ is given public key $\langle e, N\rangle$ for the RSA signature scheme and simulates the T-RSA system for $\mathcal{F}$, so that any forgery constructed by $\mathcal{F}$ without $\mathcal{F}$ guessing the password will be a forgery in the underlying RSA signature scheme.

$\mathcal{F}^*$ generates the server's key pair (pk_{svr},sk_{svr}), the user password $\pi \leftarrow_R \Pi$, and finally x and y in the normal way, except that e_1, d_1 and c are drawn randomly from $\mathbb{Z}_N$. $\mathcal{F}^*$ gives $\langle e, N\rangle$, (pk_{svr},sk_{svr}), x and y to $\mathcal{F}$.

$\mathcal{F}^*$ responds to a $f(\pi')$ query as a normal random oracle would, except that it aborts if $\pi' = \pi$ since it should output c in this case. $\mathcal{F}^*$ responds to queries to the svr, dvc, and hash oracles as in the proof of Theorem 1.

Analysis: From $\mathcal{F}$'s perspective, the above simulation is statistically indistinguishable from the real protocol, unless $\mathcal{F}$ makes a query $f(\pi)$ which occurs with probability at most $\frac{q_f}{|\Pi|}$. So, if $\mathcal{F}$ produced a forgery with probability ϵ in T-RSA, then it forges with probability $\epsilon' \approx \epsilon - \frac{q_f}{|\Pi|}$ in this simulation. To break the underlying RSA signature scheme with probability at least $\epsilon' \approx \epsilon - \frac{q_f}{|\Pi|}$, $\mathcal{F}^*$ may run $\mathcal{F}$ in this simulation and output any forgery produced by $\mathcal{F}$. □

In Theorem 4, we are given $\mathcal{F} \in \textsc{Adv}(\{\mathsf{dvc}\})$ that $(\bar{q}, \epsilon)$-breaks the T-RSA system, and then we will show that either we can construct a forger $\mathcal{F}^*$ for the underlying RSA signature scheme, or an attacker $\mathcal{A}$ against the underlying encryption scheme.

Theorem 4. *Suppose f has a negligible probability of collision over Π. If a type* $\textsc{Adv}(\{\mathsf{dvc}\})$ *forger $(\bar{q}, \epsilon)$-breaks the T-RSA system where $\epsilon = \frac{q_{\mathsf{svr}}}{|\Pi|} + \psi$, then either there exists a forger $\mathcal{F}^*$ that $(q_{\mathsf{dvc}}, \epsilon')$-breaks the underlying RSA signature scheme with $\epsilon' \approx \frac{\psi}{2}$, or there exists an attacker $\mathcal{A}$ that $(2q_{\mathsf{svr}}, \epsilon'')$-breaks the underlying encryption scheme with $\epsilon' \approx \frac{\psi}{2(q_{\mathsf{dvc}}+1)}$.*

Proof. We prove this theorem in two parts. In Part 1, we will show that if a forger $\mathcal{F}$ wins against a certain simulation of T-RSA with probability greater than $\frac{q_{\mathsf{svr}}}{|\Pi|}+\psi$, we can construct a forger $\mathcal{F}^*$ that can $(q_{\mathsf{dvc}}, \epsilon')$-break the underlying RSA signature scheme with probability $\epsilon' \approx \frac{\psi}{2}$. We say $\mathcal{F}$ wins against the simulation if $\mathcal{F}$ produces a valid forgery or if $\mathcal{F}$ makes a successful online password guess. In Part 2, assuming $\mathcal{F}$ does not win against that simulation with the probability stated above, we will show that we can construct an attacker $\mathcal{A}$ that $(2q_{\mathsf{svr}}, \epsilon'')$-breaks the underlying encryption scheme with probability $\epsilon'' \approx \frac{\psi}{2(q_{\mathsf{dvc}}+1)}$.

Part 1: $\mathcal{F}^*$ is given public key $\langle e, N\rangle$ for the RSA signature scheme and simulates the T-RSA system for $\mathcal{F}$. We say $\mathcal{F}$ makes a successful online password guess, if $\mathcal{F}$ makes a $\mathsf{serve}(s_1, y', \theta)$ query, where s_1 is a ciphertext not generated by a start query and y' is the value stored on the device, and where $\langle m, r, s_0, u\rangle \leftarrow \mathcal{D}_{sk_{\mathsf{svr}}}(s_1)$ and $s_0 == (u \cdot H(m, r))^c \bmod N$. Of course, θ must be valid.

$\mathcal{F}^*$ generates the server's key pair (pk_{svr},sk_{svr}), the user password $\pi \leftarrow_R \Pi$, and finally x and y in the normal way, except that e_1, d_1 and c are drawn randomly from $\mathbb{Z}_N$. $\mathcal{F}^*$ gives $\langle e, N\rangle$, pk_{svr}, x and $y = \mathcal{E}_{pk_{\mathsf{svr}}}(0^\ell)$ to $\mathcal{F}$. The plaintext of y is zeroed without $\mathcal{F}$ detecting it.

$\mathcal{F}^*$ responds to $\mathsf{serve}(s_1, y', \theta)$ queries as follows, if θ is valid:

- *Case 1:* (s_1, y', θ) is from a $\mathsf{start}(m)$ query: Return $\rho^{ce_1} \cdot s^{e_1 d_1} \bmod N$ as the value $(s_0)^{e_1} \bmod N$, where ρ was from the start query.

- *Case 2:* $y' == y$, but s_1 is not from a $\mathsf{start}(m)$ query: Compute $\langle m, r, s_0, u\rangle \leftarrow \mathcal{D}_{sk_{\mathsf{svr}}}(s_1)$. Also compute $s_0' \leftarrow (u \cdot H(m,r))^c \bmod N$. Abort the simulation if $s_0 == s_0'$ (as we assumed, this is a successful online password guess). Otherwise, behave like a normal server (this may have svr abort as defined in the protocol).
- *Case 3:* $y' \neq y$, but s_1 is from a $\mathsf{start}(m)$ query: Compute $\langle e_1'\rangle \leftarrow \mathcal{D}_{sk_{\mathsf{svr}}}(y')$. Invoke $\langle m, r, s_0, u\rangle$ that was computed from the start query. Then behave like a normal server (this may have svr abort as defined in the protocol).
- *Case 4:* $y' \neq y$, and s_1 is not from a $\mathsf{start}(m)$ query: Behave like a normal server (this may have svr abort as defined in the protocol).

$\mathcal{F}^*$ responds to a $\mathsf{start}(m)$ query as in Theorem 1, except setting $s_1 \leftarrow \mathcal{E}_{pk_{\mathsf{svr}}}(0^{|m|+\kappa_\Sigma+2\ell})$. $\mathcal{F}^*$ responds to a $\mathsf{finish}(s_2)$ query as in Theorem 1.

Part 1 Analysis: From $\mathcal{F}$'s perspective, the above simulation is statistically indistinguishable from the real protocol. The probability that $\mathcal{F}$ makes a successful online password guess is at most $\frac{q_{\mathsf{svr}}}{|\Pi|}$, disregarding negligible probabilities (since π was chosen randomly and f has a negligible probability of collision over Π). So, if $\mathcal{F}$ wins against the simulation with probability at least $\frac{\psi}{2} + \frac{q_{\mathsf{svr}}}{|\Pi|}$, it produces a forgery with probability at least $\frac{\psi}{2}$, and thus $\mathcal{F}^*$ produces a forgery in the underlying RSA signature scheme with probability at least $\epsilon' \approx \frac{\psi}{2}$.

Part 2: Assuming that the probability of $\mathcal{F}$ winning in Part 1 is at most $\frac{q_{\mathsf{svr}}}{|\Pi|} + \frac{\psi}{2}$, we construct an attacker $\mathcal{A}$ that breaks the underlying encryption scheme with probability $\frac{\psi}{2(q_{\mathsf{dvc}}+1)}$.

$\mathcal{A}$ is given a public key pk' from the encryption scheme, and runs two kinds of simulations of the T-RSA system for $\mathcal{F}$.

- *SIM 1:* A simulator gives pk' to $\mathcal{F}$ as the server's public encryption key, and then simulates T-RSA exactly, but using a decryption oracle to decrypt messages encrypted under key pk' by the adversary (correct θ and signature key pairs are postulated). There will be at most $2q_{\mathsf{svr}}$ of these. Note that the decryptions of y and any s_1 generated by the dvc would already be known to the simulator. This simulation would be perfectly indistinguishable from the real protocol to $\mathcal{F}$.
- *SIM 2:* The same simulation is done, but with y and all s_1 values generated by the device changed to encryptions of strings of zeros. The server pretends the encryptions are done in the normal ways.

Part 2 Analysis: SIM 2 is statistically equivalent to the Part 1 simulation, except for aborting on a successful online password guess. So, as we assumed, the probability of $\mathcal{F}$ forging in SIM 2 is at most $\frac{q_{\mathsf{svr}}}{|\Pi|} + \frac{\psi}{2}$, while the probability of $\mathcal{F}$ forging SIM 1 is at least $\frac{q_{\mathsf{svr}}}{|\Pi|} + \psi$.

Let us utilize a standard hybrid argument to construct $\mathcal{A}$ [24, 31]. Let experiment $j \in \{0, \cdots, q_{\mathsf{dvc}} + 1\}$ correspond to the first j ciphertexts, generated by $\mathcal{A}$, being of the normal messages, and the remainder be encryptions of strings of 0's. Let p_j be the probability of $\mathcal{F}$ forging in experiment j. Then the average value

for $i \in \{0, \cdots, q_{\mathsf{dvc}}\}$ of $p_{i+1} - p_i$ is at least $\frac{\psi}{2} \cdot \frac{1}{(q_{\mathsf{dvc}}+1)}$. In order to construct $\mathcal{A}$, we may have $\mathcal{A}$ choose a random value $i \in \{0, \cdots, q_{\mathsf{dvc}}\}$, and run experiment i as above. Subsequently, we make $\mathcal{A}$ call the test oracle for the $(i+1)^{\text{st}}$ encryption to be generated by the simulator, where the two messages X_0 and X_1 submitted to the test oracle are the normal message and the string of zeros, respectively. Then $\mathcal{A}$ outputs 0 if $\mathcal{F}$ forges, and 1 otherwise (since $\mathcal{F}$ could have noticed the zeroed messages and distinguished the simulation). By the analysis above, $\mathcal{A}$ breaks the underlying encryption scheme with probability $\frac{\psi}{2(q_{\mathsf{dvc}}+1)}$, disregarding negligible probabilities. □

An adversary in class $\textsc{Adv}(\{\mathsf{dvc}, \pi\})$ must be handled in a slightly different manner because it is capable of having exact c. So, in Theorem 5, we assume T-RSA′ of which the underlying RSA scheme is deterministic [10, 31]. The deterministic value r is regarded for consistency.

$$T'_{\langle d,N\rangle}(m)\text{: } r \leftarrow_D \{0,1\}^{\kappa_{\Sigma'}};\ s \leftarrow (H(m,r))^d \bmod N;\ \text{return } \langle s, r\rangle$$

Also we call RSA′ as the underlying blind RSA scheme [12] for s_0.

$$RSA'_{\langle d,N\rangle}(m,u)\text{: } r \leftarrow_D \{0,1\}^{\kappa_{\Sigma'}};\ s \leftarrow (u \cdot H(m,r))^d \bmod N;\ \text{return } \langle s, r\rangle$$

Theorem 5. *If a type* $\textsc{Adv}(\{\mathsf{dvc}, \pi\})$ *forger* $(\bar{q}, \epsilon)$*-breaks the T-RSA′ system, then either there exists a forger* $\mathcal{F}^*$ *that* $(2q_{\mathsf{svr}}, \epsilon'')$*-breaks the underlying RSA signature scheme with* $\epsilon'' \approx \frac{\epsilon}{2}$*, or there exists an attacker* $\mathcal{A}$ *that* $(2q_{\mathsf{svr}}, \epsilon'')$*-breaks the underlying encryption scheme with* $\epsilon''' \approx \frac{\epsilon}{2(q_{\mathsf{dvc}}+1)}$.

Proof. We prove this theorem in two parts. In Part 1, we show that if a forger $\mathcal{F}$ wins against a certain simulation of T-RSA′ with probability greater than ϵ, we can construct a forger $\mathcal{F}^*$ that can $(2q_{\mathsf{svr}}, \epsilon')$-break the underlying RSA′ blind signature scheme with $\epsilon' \approx \frac{\epsilon}{2}$, and also $(2q_{\mathsf{svr}}, \epsilon'')$-break the underlying RSA signature scheme with $\epsilon'' = \epsilon'$. In Part 2, assuming $\mathcal{F}$ does not forge a signature in that simulation with the probability stated above, we show that we can construct an attacker $\mathcal{A}$ that $(2q_{\mathsf{svr}}, \epsilon''')$-breaks the underlying encryption scheme with $\epsilon''' \approx \frac{\epsilon}{2(q_{\mathsf{dvc}}+1)}$.

Part 1: $\mathcal{F}^*$ is given public key $\langle e, N\rangle$ for the RSA signature scheme and simulates the T-RSA′ system for $\mathcal{F}$. $\mathcal{F}^*$ generates the server's key pair $(pk_{\mathsf{svr}}, sk_{\mathsf{svr}})$, the user password $\pi \leftarrow_R \Pi$, and finally x and y in the normal way, except that e_1, d_1 and c are drawn randomly from $\mathbb{Z}_N$. $\mathcal{F}^*$ gives $\langle e, N\rangle$, pk_{svr}, π, x and $y = \mathcal{E}_{pk_{\mathsf{svr}}}(0^\ell)$ to $\mathcal{F}$. The plaintext of y is zeroed without $\mathcal{F}$ detecting it.

$\mathcal{F}^*$ responds to $\mathsf{serve}(s_1, y', \theta)$ queries as follows, if θ is valid:

- *Case 1:* (s_1, y', θ) is from a $\mathsf{start}(m)$ query: Query the blind signature oracle with $\langle m, u\rangle$ to get $\langle s_2, r\rangle$ where u is from the $\mathsf{start}(m)$ query (note that r must be the same as r from the $\mathsf{start}(m)$ query due to its deterministic property). Return s_2.
- *Case 2:* $y' == y$, but s_1 is not from a $\mathsf{start}(m)$ query: Compute $\langle m, r, s_0, u\rangle \leftarrow \mathcal{D}_{sk_{\mathsf{svr}}}(s_1)$. Also compute $s'_0 \leftarrow (u \cdot H(m,r))^c \bmod N$. If $s_0 == s'_0$, query the

blind signature oracle to get $\langle s_2, r\rangle$ and return s_2 (again we could disregard r). Otherwise, behave like a normal server (this may have svr abort as defined in the protocol).

- *Case 3:* $y' \neq y$, but s_1 is from a $\mathsf{start}(m)$ query: Compute $\langle e_1'\rangle \leftarrow \mathcal{D}_{sk_{\mathsf{svr}}}(y')$. Invoke $\langle m, r, s_0, u\rangle$ that was computed from the start query. Then behave like a normal server (this may have svr abort as defined in the protocol).
- *Case 4:* $y' \neq y$, and s_1 is not from a $\mathsf{start}(m)$ query: Behave like a normal server (this may have svr abort as defined in the protocol).

$\mathcal{F}^*$ responds to a $\mathsf{start}(m)$ query as in the real protocol with regard to the definition of T-RSA′, except setting $s_1 \leftarrow \mathcal{E}_{pk_{\mathsf{svr}}}(0^{|m|+\kappa_\Sigma+2\ell})$. For a $\mathsf{finish}(s_2)$ query corresponding to a $\mathsf{start}(m)$ query, $\mathcal{F}^*$ responds as in the real protocol with regard to the definition of T-RSA′ if s_2 was from a corresponding $\mathsf{serve}(s_1, y, \theta)$ query. Otherwise, it has dvc abort.

Part 1 Analysis: From $\mathcal{F}$'s perspective, the above simulation is statistically indistinguishable from the real protocol. If $\mathcal{F}$ forges in the simulation with probability at least $\frac{\epsilon}{2}$, it produces a forgery with probability at least $\frac{\epsilon}{2}$, and thus $\mathcal{F}^*$ produces a forgery in the underlying RSA′ blind signature scheme with probability at least $\epsilon' \approx \frac{\epsilon}{2}$, by simply running F in the simulation above and finally outputs the result with $w \cdot s \bmod N$ where $w \leftarrow_R \{0,1\}^\ell$ and s is a forgery of $\mathcal{F}$. In that sense, $\mathcal{F}^*$ can produce a forgery in the underlying RSA signature scheme with probability $\epsilon'' = \epsilon'$, by simply outputting $\langle s, r\rangle$ where s is still the forgery of $\mathcal{F}$. Note that querying the corresponding signature oracle can be made on time in the simulation or afterward with the transcripts.

Part 2: Assuming that the probability of $\mathcal{F}$ winning in Part 1 is at most $\frac{\epsilon}{2}$, we construct an attacker $\mathcal{A}$ that breaks the underlying encryption scheme with probability $\frac{\epsilon}{2(q_{\mathsf{dvc}}+1)}$.

$\mathcal{A}$ is given a public key pk' from the encryption scheme, and runs two kinds of simulations of the T-RSA′ system for $\mathcal{F}$.

- *SIM 1:* A simulator gives pk' to $\mathcal{F}$ as the server's public encryption key, and then simulates T-RSA exactly, but using a decryption oracle to decrypt messages encrypted under key pk' by the adversary (correct θ and signature key pairs are postulated). There will be at most $2q_{\mathsf{svr}}$ of these. Note that the decryptions of y and any s_1 generated by the dvc would already be known to the simulator. This simulation would be perfectly indistinguishable from the real protocol to $\mathcal{F}$.
- *SIM 2:* The same simulation is done, but with y and all s_1 values generated by the device changed to encryptions of strings of zeros. The server pretends the encryptions are done in the normal ways.

Part 2 Analysis: SIM 2 is statistically equivalent to the Part 1 simulation. Still, the probability of $\mathcal{F}$ forging in SIM 2 is at most $\frac{\epsilon}{2}$, while the probability of $\mathcal{F}$ forging SIM 1 is at least ϵ.

We use the standard hybrid argument to construct $\mathcal{A}$ [24, 31]. Let experiment $j \in \{0, \cdots, q_{\mathsf{dvc}} + 1\}$ correspond to the first j ciphertexts, generated by $\mathcal{A}$, being

of the normal messages, and the remainder be encryptions of strings of 0's. Let p_j be the probability of $\mathcal{F}$ forging in experiment j. Then the average value for $i \in \{0, \cdots, q_{\mathsf{dvc}}\}$ of $p_{i+1} - p_i$ is at least $\frac{\psi}{2} \cdot \frac{1}{(q_{\mathsf{dvc}}+1)}$. In order to construct $\mathcal{A}$, we may have $\mathcal{A}$ choose a random value $i \in \{0, \cdots, q_{\mathsf{dvc}}\}$, and run experiment i as above. Subsequently, we make $\mathcal{A}$ call the test oracle for the $(i+1)^{\text{st}}$ encryption to be generated by the simulator, where the two messages X_0 and X_1 submitted to the test oracle are the normal message and the string of zeros, respectively. Then $\mathcal{A}$ outputs 0 if $\mathcal{F}$ forges, and 1 otherwise (since $\mathcal{F}$ could have noticed the zeroed messages and distinguished the simulation). By the analysis above, $\mathcal{A}$ breaks the underlying encryption scheme with probability $\frac{\psi}{2(q_{\mathsf{dvc}}+1)}$, disregarding negligible probabilities. □