DIMACS Technical Report 2001-19
May 2001

Networked Cryptographic Devices Resilient to Capture

by

Philip MacKenzie[1] Michael K. Reiter[2]

Bell Labs, Lucent Technologies, Murray Hill, New Jersey, USA
{philmac,reiter}@research.bell-labs.com

[1]Permanent Member
[2]Permanent Member

DIMACS is a partnership of Rutgers University, Princeton University, AT&T Labs-Research, Bell Labs, NEC Research Institute and Telcordia Technologies (formerly Bellcore).

DIMACS was founded as an NSF Science and Technology Center, and also receives support from the New Jersey Commission on Science and Technology.

ABSTRACT

We present a simple technique by which a device that performs private key operations (signatures or decryptions) in networked applications, and whose local private key is activated with a password or PIN, can be immunized to offline dictionary attacks in case the device is captured. Our techniques do not assume tamper resistance of the device, but rather exploit the networked nature of the device, in that the device's private key operations are performed using a simple interaction with a remote server. This server, however, is untrusted—its compromise does not reduce the security of the device's private key unless the device is also captured—and need not have a prior relationship with the device. We further extend this approach with support for *key disabling*, by which the rightful owner of a stolen device can disable the device's private key even if the attacker already knows the user's password.

1 Introduction

A device that performs signatures or decryptions using the private key of a public key pair, and that stores the private key locally on stable storage, is typically vulnerable to exposure of that private key if the device is captured. While encryption of the private key under a password is common, the ease with which passwords succumb to offline dictionary attacks (e.g., see [36, 28, 18, 43]) implies that better protections are needed. Many such protections have been proposed, but most require tamper-resistance of the device. Others used in practice replace the password with a stronger key stored on another device that the user holds, thus moving the burden of protection to that device.

In this paper we propose a simple, software-only technique to render the private key of a networked device invulnerable to offline dictionary attacks, even if the device is captured. Our technique exploits the fact that the device has network connectivity at the time it is required to perform a private key operation, and thus can interact with a remote party at that time to complete the operation. This is characteristic of virtually any device involved in an interactive authentication or key exchange protocol.

The way in which we exploit network connectivity is to postulate a remote server that assists the device in performing its private key operation. This remote server need not have any preexisting relationship with, or knowledge of, the device (though the device needs a public key for the server). Moreover, the server is untrusted: we prove that the server, even if it misbehaves, gains no information that would help it to compute signatures that verify with the device's public key or to decrypt messages encrypted under the device's public key. The only behavior that we require of the server is that it execute the correct protocol to respond to a well-formed request, and that it stop responding to invocations pertaining to a device's public key (perhaps for a period of time) after it has received sufficiently many malformed requests associated with this public key. This latter behavior is required to prevent an *online* dictionary attack against the password. We note, however, that this feature does not present a denial-of-service vulnerability, since in our protocol, an attacker can conduct an online dictionary attack only after it has captured the device—and so use of the device by the legitimate user is presumably already denied.

We present two types of protocols that achieve the above properties. These types functionally differ on whether they enable the device's private key to be *disabled*. If the device is stolen, it is natural for the device's rightful owner to wish to disable the use of the private key, to account for the possibility that the attacker already knows the user's password (e.g., by observing the user type it) or can guess it in very few tries (e.g., due to his intimate knowledge of the user). In one type of protocol we present, the user can issue a request to the server to disable future use of the private key associated with the device's public key. Once the server receives this request and verifies it is well-formed, the device's key is rendered (provably) useless to the attacker, even if the attacker knows the user's password. The attacker will thus be unable to employ the key in future interactive protocols or to decrypt future encrypted messages. This feature is especially useful if revocation of the device's public key via a public key infrastructure (e.g., a certificate revocation list) has

an associated delay (if it exists at all); in contrast, using our system the private key can be disabled immediately.

The ability to disable a private key seems to come at a cost in terms of compatibility with existing protocols. Our protocol without this feature is compatible with any public key cryptosystem or signature scheme in use by the device, and any protocol using them. In contrast, our protocols supporting key disabling are dependent on the type of private key operations in use; here we give protocols for RSA [40] signatures and ElGamal [17] decryption. These easily generalize to many other signature and decryption protocols. In addition, to achieve provable security, our signature protocols supporting key disabling expose the message being signed to the server. As such, it is compatible only with applications that sign public data. This is consistent with, e.g., TLS 1.0 [15], but is incompatible with protocols that sign private data before encrypting it. There are variations of our RSA signature protocol, for example, that do not require the message to be disclosed to the server, but proving them secure requires nonstandard assumptions about the security of RSA.

2 Prior work

The work of which we are aware that bears most conceptual similarity to our own is that of Yaksha [20], a public key extension to Kerberos. Yaksha shares our goals of achieving password-protected private key operations that are not vulnerable to offline dictionary attacks, and the user's device employs a remote server to achieve this protection. However, there are a number of important differences between our work and Yaksha. First, Yaksha supports only RSA private key operations. In contrast, here we present protocols for any type of private key operations (generically), for RSA signatures, and for ElGamal decryption; also other private key operations can be accommodated within our framework, including DSA signatures, as a corollary of [33]. Second, Yaksha does not provide security against server compromise; i.e., the Yaksha server must be trusted, whereas ours need not be. Third, the Yaksha server requires initialization per user, whereas our server does not. These latter two properties make the server component of our solution more suitable to a service-provider offering, for example.

More distantly related work is [25]. This work proposes methods to encrypt a DSA or RSA private key using a password so that guesses at the password cannot be verified by an attacker who captures the device holding that private key. This feature comes at a severe price, however. For example, the device's "public" key must be kept secret, even from the device itself: obviously if the attacker learns the public key, then he can verify a successfully decrypted private key. So, the public key must be hidden from all but a few trusted servers that verify signatures produced by the device or encrypt messages for the device. And, it is essential that no verifiable plaintext be encrypted, since this, too, could be used to verify guesses at the password. In contrast, our work achieves similar goals without imposing such awkward system constraints. Our solutions require nothing of the system surrounding the device other than the ability for the device to communicate over a network when it performs private key operations.

One way to partially reach our goals is to simply not store the device's private key on the device, but rather have the device download it from the server when needed (e.g., [37]). Indeed, one of our protocols somewhat resembles this approach. To ensure that the private key is downloaded only to the user's device, the device first proves it has been given the user's password. For this purpose there are numerous published protocols by which the device can authenticate to and exchange a key with a server using a password input by its user, without exposing that password to offline dictionary attacks. Some protocols require the device to already have a public key for the server (e.g., [31, 24, 19]), others do not (e.g., [9, 27, 42, 5, 11, 32]). Since the device stores at most only public information, its capture is of no consequence. On the other hand, in all of these protocols, the server either knows the user's password or else can mount an offline dictionary attack against it. More importantly, when these protocols are used for the retrieval of a private key from the server, the private key (which would most likely be encrypted with the password) would be exposed to the server after a successful offline dictionary attack on the password. Recent proposals resort to multiple servers and require that at most some threshold cooperate in a dictionary attack [19], but nevertheless this remains a differentiator of our approach: our server is entirely untrusted. A second differentiator of our work is that prior work does not permit key disabling to address the possibility that an attacker already knows the user's password or guesses it quickly: once the attacker guesses the password and downloads the private key, the attacker can use it for an unlimited time. In contrast, we present protocols in which the private key can be disabled, even *after* the attacker has captured the user's device *and* guessed the user's password.

Short of rendering the device's private key invulnerable to an offline dictionary attack once the device is captured, perhaps the next best thing is to ensure that the private key cannot be used to sign messages dated before the device was captured. This is achieved by *forward secure* signature schemes, which intuitively change the private key (but not the public key) over time so that the captured private key can be used to sign messages only dated in the future (e.g., [4, 30]). If the device can sense that its private key is about to be discovered, as might be possible if the device is a coprocessor with tamper detection circuitry, then another alternative is for the device to change the private key when it detects a pending compromise so that future signatures subliminally disclose to an authority receiving those signatures that the device has been compromised [23]. In contrast to these approaches, our goal is to prevent any future signatures by the attacker once the device is captured, rather than permitting them in a limited way (as forward secure signature schemes do) or in a way that subliminally alerts an authority (as in [23]).

Finally, our use of a server to assist the device in performing signatures or decryptions is reminiscent of *server aided protocols*, whereby the computational burden of a secret cryptographic computation is moved from the device to a more powerful server. Some of these protocols place trust in the server and thus expose the device's private information to it (e.g., [1, 16]), while others attempt to hide the private key from the server but nevertheless have the server do the bulk of the computation (e.g., [34, 2, 26]). Our work differs in its goals: our intention is to render the device impervious to an offline dictionary attack once

captured, rather than to reduce the computation required of the device. On the contrary, in our protocols, the device ends up performing at least as much computation as it would if it were to perform the secret computation entirely itself. While it seems fairly straightforward to combine our protocols with some of these techniques, doing so while maintaining provable security looks to be a challenge.

3 Preliminaries

In this section we informally state the goals for our systems. We also introduce preliminary definitions and notation that will be necessary for the balance of the paper.

3.1 Goals

We presume a system with a device dvc and a server svr that communicate by exchanging messages over a public network. In our protocols, the device is used either for generating signatures or decrypting messages, and does so by interacting with the server. The signature or decryption operation is password-protected, by a password π_0. The system is initialized with public data, secret data for the device, secret data for the user of the device (i.e., π_0), and secret data for the server. The public and secret data associated with the server should simply be a certified public key and associated private key, which most likely would be set up well before the device is initialized. The device-server protocol allows a device operated by a legitimate user (i.e., one who knows π_0) to sign or decrypt a message with respect to the public key of the device, after communicating with the server. In those systems supporting key disabling, device initialization may create additional secret data that, if sent to the server, will cause the server to no longer execute the decryption or signing protocol with that device.

Each adversary we consider is presumed to control the network; i.e., the attacker controls any inputs to dvc or svr, and observes their outputs. Moreover, an adversary can "capture" certain resources. The possible resources that may be captured by the attacker are dvc, svr, and π_0. Once captured, the entire static contents of the resource become known to the attacker. The one restriction on the adversary is that if he captures dvc, then he does so after dvc initialization and while dvc is in an inactive state—i.e., dvc is not presently executing the protocol with π_0 as input—and that π_0 is not subsequently input to the device by the user. This decouples the capture of dvc and π_0, and is consistent with our motivation that dvc is captured while not in use by the user and, once captured, is unavailable to the user.

We denote by $\textsc{Adv}(S)$, where $S \subseteq \{\mathsf{dvc}, \mathsf{svr}, \pi_0\}$, the class of adversaries who succeed in capturing the elements of S. As such, $\textsc{Adv}(S_1) \subseteq \textsc{Adv}(S_2)$ if $S_1 \subseteq S_2$. The security goals of our systems are informally stated as follows:

I. Any adversary in $\textsc{Adv}(\{\mathsf{svr}, \pi_0\})$ is unable to forge signatures or decrypt messages for the device (with overwhelming probability).

II. Any adversary in $\text{ADV}(\{\mathsf{dvc}\})$ can forge signatures or decrypt messages for the device with probability at most $q/|\mathcal{D}|$ after q invocations of the server, where $\mathcal{D}$ is the space from which the user's password is drawn (uniformly at random).

III. Any adversary in $\text{ADV}(\{\mathsf{dvc}, \mathsf{svr}\})$ can forge signatures or decrypt messages for the device only if it succeeds in an offline dictionary attack on the user's password.

IV. Any adversary in $\text{ADV}(\{\mathsf{dvc}, \pi_0\})$ can forge signatures or decrypt messages for the device only until the device key is disabled (in those systems supporting key disabling), and subsequently cannot forge signatures or decrypt messages for the device.

3.2 Definitions

In order to state our protocols to meet the goals outlined in Section 3.1, we first introduce some definitions and notation.

Security parameters Let κ be the main cryptographic security parameter; a reasonable value today may be $\kappa = 160$. We will use $\lambda > \kappa$ as a secondary security parameter for public keys. For instance, in an RSA public key scheme may we may set $\lambda = 1024$ to indicate that we use 1024-bit moduli.

Hash functions We use h, with an additional subscript as needed, to denote a hash function. Unless otherwise stated, the range of a hash function is $\{0,1\}^\kappa$.

We do not specify here the exact security properties (e.g., one-wayness, collision resistance, or pseudorandomness) we will need for the hash functions (or keyed hash functions, below) that we use. To formally prove that our protocols meet every goal outlined above, we generally require that these hash functions behave like random oracles [6]. (For heuristics on instantiating random oracles, see [6].) However, for certain subsets of goals, weaker properties may suffice; details will be given in the individual cases.

Keyed hash functions A keyed hash function family is a family of hash functions $\{f_v\}$ parameterized by a secret value v. We will typically write $f_v(m)$ as $f(v,m)$, as this will be convenient in our proofs. In this paper we employ various keyed hash functions with different ranges, which we will specify when not clear from context.

We will also use a specific type of keyed hash function, a message authentication code (MAC). We denote a MAC family as $\{\mathsf{mac}_a\}$. In this paper we do not require MACs to behave like random oracles, but to have the following standard property: If a is unknown, then given zero or more pairs $<m_i, \mathsf{mac}_a(m_i)>$, it is computationally infeasible to compute any pair $<m, \mathsf{mac}_a(m)>$ for any new $m \neq m_i$.

Encryption schemes An *encryption scheme* $\mathcal{E}$ is a triple (G_{enc}, E, D) of algorithms, the first two being probabilistic, and all running in expected polynomial time. G_{enc} takes as input 1^λ and outputs a public key pair (pk, sk), i.e., $(pk, sk) \leftarrow G_{enc}(1^\lambda)$. E takes a public key pk and a message m as input and outputs an encryption c for m; we denote this $c \leftarrow E_{pk}(m)$. D takes a ciphertext c and a secret key sk as input and returns either a message m such that c is a valid encryption of m, if such an m exists, and otherwise returns $\perp$. Our protocols require an encryption scheme secure against adaptive chosen ciphertext attacks [39]. Practical examples can be found in [7, 13].

Signature schemes A *digital signature scheme* $\mathcal{S}$ is a triple (G_{sig}, S, V) of algorithms, the first two being probabilistic, and all running in expected polynomial time. G_{sig} takes as input 1^λ and outputs a public key pair (pk, sk), i.e., $(pk, sk) \leftarrow G_{sig}(1^\lambda)$. S takes a message m and a secret key sk as input and outputs a signature σ for m, i.e., $\sigma \leftarrow S_{sk}(m)$. V takes a message m, a public key pk, and a candidate signature σ' for m as input and returns the bit $b = 1$ if σ' is a valid signature for m, and otherwise returns the bit $b = 0$. That is, $b \leftarrow V_{pk}(m, \sigma')$. Naturally, if $\sigma \leftarrow S_{sk}(m)$, then $V_{pk}(m, \sigma) = 1$.

We say a signature scheme is *matchable* if for each public key pk produced by $G_{sig}(1^\lambda)$ there is a single secret key sk that would be produced (i.e., the probability of $(pk, sk) \leftarrow G_{sig}(1^\lambda)$ and $(pk, sk') \leftarrow G_{sig}(1^\lambda)$ with $sk \neq sk'$ is zero), and there is a probabilistic algorithm M that runs in expected polynomial time and that takes as input a public key pk and a secret key sk, and returns 1 if sk is the single private key corresponding to pk (i.e., if $G_{sig}(1^\lambda)$ could have produced (pk, sk) with non-zero probability) and returns 0 otherwise. In most popular signature schemes, including those we consider here, there is a straightforward way to implement the M function. (We can define *matchable* encryption schemes similarly.)

4 A simple protocol without key disabling

We begin by presenting a simple protocol for achieving goals I, II, and III described in Section 3.1. Since this protocol remains the same regardless of whether the device is used to decrypt or sign, here we discuss the protocol using terminology as if the device is used for signing. This system is parameterized by the device's signature scheme $\mathcal{S}$ and an encryption scheme $\mathcal{E}$ for the server,[1] and works independently of the form of $\mathcal{S}$ and $\mathcal{E}$. We thus refer to this protocol as "generic", and denote the protocol by GENERIC.

The intuition behind GENERIC is exceedingly simple. At device initialization time, the private key of the device is encrypted in a way that can be recovered only with the cooperation of both the device (if it is given the user's password) and the server. This ciphertext, called a *ticket*, also embeds other information that enables the server to authenticate requests that accompany the ticket as coming from a device that has been given the user's password. When the device is required to perform an operation with its private key, it sends the ticket

[1] When speaking about security of this and later protocols against offline dictionary attack, we also include a parameter $\mathcal{D}$ to denote a dictionary of the possible passwords.

to the server. The device accompanies the ticket with evidence of its knowledge of the user's password; the server can check this evidence against information in the ticket. The server then performs a transformation on the ticket to "partially decrypt" it, and returns the result to the device. The device completes the decryption to recover its private key. The device may then use the private key for performing the required operations, and may even cache the key in volatile memory for some period of time so that additional operations can be performed without contacting the server for each one.

Note that a protocol of this form cannot support key disabling: if an attacker captures the device and guesses the user's password (i.e., the adversary is in $\textsc{Adv}(\{\mathsf{dvc}, \pi_0\})$), then it can retrieve the private key and keep it forever. Limiting the damage an attacker can do in this case requires assistance from some external mechanism for revoking the device's public key, if such a mechanism exists.

In the following two sections, we detail the steps of the initialization algorithm and the key retrieval protocol. In Section 6 this system is formally proven (in the random oracle model) to meet goals I–III of Section 3.1.

4.1 Device initialization

The inputs to device initialization are the server's public encryption key pk_{svr}, the user's password π_0, the device's public signature verification key pk_{dvc}, and the corresponding private signing key sk_{dvc}. The steps of the initialization algorithm proceed as follows, where "$z \leftarrow_R S$" is used to denote assignment to z of an element of S selected uniformly at random.

$$
\begin{aligned}
v &\leftarrow_R \{0,1\}^\kappa \\
a &\leftarrow_R \{0,1\}^\kappa \\
b &\leftarrow h(\pi_0) \\
c &\leftarrow f(v, \pi_0) \oplus sk_{\mathsf{dvc}} \\
\tau &\leftarrow E_{pk_{\mathsf{svr}}}(<a, b, c>)
\end{aligned}
$$

The values v, a, τ, pk_{dvc}, and pk_{svr} are saved in stable storage on the device. All other values, including sk_{dvc}, π_0, b and c, are deleted from the device. We assume that f outputs a value of length equal to the length of sk_{dvc}. For the protocol of Section 4.2, we assume this length is λ.

The value τ is the "ticket" to which we referred previously. Note that this ticket encapsulates a value c from which the device can recover sk_{dvc} with knowledge of the user's password. The server's role in the key retrieval protocol will thus involve decrypting this ticket and sending c to the device (encrypted). Note that c does not provide the basis for the server to mount an attack against sk_{dvc}, since the server does not know v.

4.2 Key retrieval protocol

The input provided to the device to initiate the key retrieval protocol is the input password π and all of the values saved on stable storage in the initialization protocol of Section 4.1. The protocol by which the device retrieves sk_{dvc} is shown in Figure 1.

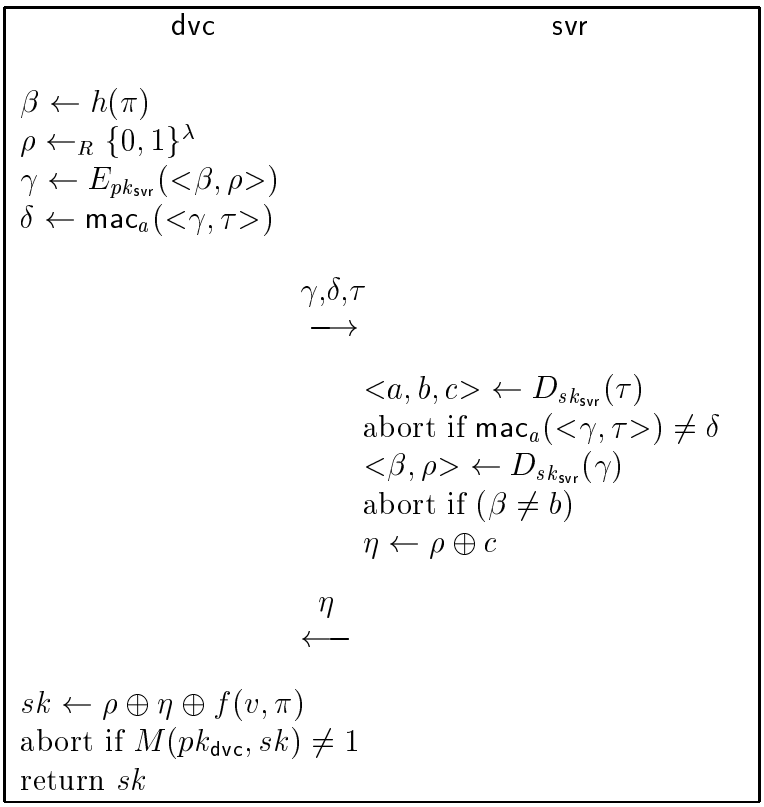

Figure 1: GENERIC key retrieval protocol

In Figure 1, β is an authenticator that proves knowledge of π to the server. ρ acts as a one-time pad by which the server encrypts c to return it to the device. γ is an encryption of β and ρ to securely transport them to the server. The value δ is a message authentication code that is generated from the MAC key a stored on the device, and that the server uses to confirm that this request actually originated from the device. Though δ is not required to prove security of this protocol, it nevertheless is important in practice: it enables the server to distinguish requests bearing τ but not originating from the device (i.e., $\mathsf{mac}_a(<\gamma, \tau>) \neq \delta$), from requests bearing τ that originate from the device but for which the device's knowledge of the user's password cannot be verified (i.e., $\beta \neq b$). The latter category may indicate an online dictionary attack, and accordingly the ticket τ should be ignored (perhaps for some period of time) after sufficiently many such requests. The former type should not "count against" τ, however, since they do not pose a risk to the password; indeed, the authenticator β is never checked in these cases. On the contrary, if this former category were treated like the latter, then this would enable a denial-of-service attack on τ (i.e., the device) in which an attacker, having seen τ pass on the network, submits requests to the server containing τ and random values for γ and δ.

It is important for security that the device delete β, ρ and, of course, sk when it is done with them, so that none of these values are available to an attacker who subsequently

captures the device. In particular, these values should never be stored on stable storage on the device to ensure, e.g., that they will disappear from the device if the device crashes.

Brief intuition for the security of this protocol is as follows. First, goal I is achieved due to the encryption of sk_{dvc} by $f(v, \pi_0)$, since an adversary in $\textsc{Adv}(\{\mathsf{svr}, \pi_0\})$ does not know v. Goal II is achieved since the only way an adversary in $\textsc{Adv}(\{\mathsf{dvc}\})$ gains information about the password is by submitting guesses at β (or rather, β's resulting from guesses at the password) to the server. Finally, even an adversary in $\textsc{Adv}(\{\mathsf{dvc}, \mathsf{svr}\})$ is required to conduct an offline dictionary attack against the password to discover sk_{dvc}, since sk_{dvc} is encrypted using $f(v, \pi_0)$.

5 Systems supporting key disabling

In this section we present protocols that satisfy all of the goals of Section 3.1, including the ability for the user to *disable* the private key of the device even after the attacker has captured the device *and* guessed the user's password. As described in Section 4, the reason that key disabling is not possible with GENERIC is that the device's private key is recovered by the device as part of that protocol. As a result, an attacker who captures the device and guesses the user's password can recover the private key and use it indefinitely.

In order to make key disabling possible, we thus design protocols in which the private key is never recovered by the device. Rather, the device performs each signature or decryption operation individually by interacting with the server. This is achieved by 2-out-of-2 function sharing, where the function being shared is the device's signature or decryption function. More precisely, when the device is initialized, two *shares* of the device's private key are generated. The first share is constructed so that it can be generated from the user's password and information stored on the device. The second share, plus other data for authenticating requests from the device, are encrypted under pk_{svr} to form the device's ticket. Both shares are then deleted from the device. In the device's signature or decryption protocol, the device sends its ticket plus evidence that it was given the user's password, the server verifies this using information in the ticket, and then the server contributes its portion of the computation using its share. Together with the device's contribution using its share (generated from the user's password), the signature or decryption can be formed.

Disabling the private key sk_{dvc} can be achieved by requesting that the server permanently ignore the device's ticket. Once this is done, further queries by the attacker—specifically, any adversary in $\textsc{Adv}(\{\mathsf{dvc}, \pi_0\})$—will not yield further signatures or decryptions. Of course, to prevent a denial-of-service attack against the device even without it being stolen, requests to disable the device's ticket must be authenticated; our protocols achieve this, too. Our protocols *provably* meet all of the goals stated in Section 3.1 in the random oracle model.

The feature of key disabling apparently comes with costs in terms of compatibility with existing protocols. For example, in the signature protocol we demonstrate here, the server learns the message m being signed. It is therefore important that m be public information if the server is untrusted. This requirement is consistent with signatures in TLS 1.0 [15], for example, since in that protocol, parties sign only public information. However, it may be

inconsistent with other protocols that encrypt private information *after* signing it. Second, due to our use of function sharing in these protocols, they are generally dependent on the particular signature or decryption algorithm in use. In the following subsections, we describe protocols for RSA signatures and ElGamal decryption, though our techniques also generalize to many other signature and decryption schemes.

5.1 S-RSA: a protocol for RSA signatures

In this section we suppose the device signs using a standard encode-then-sign RSA signature algorithm (e.g., "hash-and-sign" [14]) as described below. Accordingly, we refer to this protocol as S-RSA. The public key of the device is $pk_{\mathsf{dvc}} = <e, N>$ and the secret key is $sk_{\mathsf{dvc}} = <d, N, \phi(N)>$, where $ed \equiv_{\phi(N)} 1$, N is the product of two large prime numbers, and ϕ is the Euler totient function. (The notation $\equiv_{\phi(N)}$ means equivalence modulo $\phi(N)$.) The device's signature on a message m is defined as follows, where encode is the encoding function associated with S, and κ_{sig} denotes the number of random bits used in the encoding function (e.g., $\kappa_{sig} = 0$ for a deterministic encoding function):

$S_{<d,N,\phi(N)>}(m)$: $r \leftarrow_R \{0,1\}^{\kappa_{sig}}$
$s \leftarrow (\mathsf{encode}(m, r))^d \bmod N$
return $<s, r>$

Here, the signature is $\sigma = <s, r>$, though it may not be necessary to include r if it can be determined from m and s. We remark that "hash-and-sign" is an example of this type of signature in which the encoding function is simply a (deterministic) hash of m, and that PSS [8] is another example of this type of signature with a probabilistic encoding. Both of these types of signatures were proven secure against adaptive chosen message attacks in the random oracle model [6, 8]. Naturally any signature of this form can be verified by checking that $s^e \equiv_N \mathsf{encode}(m, r)$. In the function sharing primitive used in our protocol, d is broken into shares d_1 and d_2 such that $d_1 + d_2 \equiv_{\phi(N)} d$ [10].

In the following three sections, we detail the steps of the initialization algorithm, the signing protocol, and the key disabling protocol. In Section 7 this system is formally proven (in the random oracle model) to meet goals I–IV of Section 3.1.

5.1.1 Device initialization

The inputs to device initialization are the server's public encryption key pk_{svr}, the user's password π_0, the device's public key $pk_{\mathsf{dvc}} = <e, N>$, and the corresponding private key $sk_{\mathsf{dvc}} = <d, N, \phi(N)>$. The initialization algorithm proceeds as follows:

$t \leftarrow_R \{0,1\}^\kappa$
$u \leftarrow h_{\mathsf{dsbl}}(t)$
$v \leftarrow_R \{0,1\}^\kappa$
$a \leftarrow_R \{0,1\}^\kappa$
$b \leftarrow h(\pi_0)$
$d_1 \leftarrow f(v, \pi_0)$
$d_2 \leftarrow d - d_1 \bmod \phi(N)$
$\tau \leftarrow E_{pk_{\mathsf{svr}}}(<a, b, u, d_2, N>)$

Here, we assume that f outputs an element of $\{0,1\}^{\lambda+\kappa}$. The values t, v, a, τ, pk_{dvc}, and pk_{svr} are saved on stable storage in the device. All other values, including u, b, d, d_1, d_2, $\phi(N)$, and π_0, are deleted from the device. The values t and τ should be backed up offline for use in disabling if the need arises. The value τ is the device's "ticket" that it uses to access the server.

5.1.2 Signature protocol

Here we present the protocol by which the device signs a message m. The input provided to the device for this protocol is the input password π, the message m, and all of the values saved on stable storage in the initialization protocol of Section 5.1.1. The protocol is described in Figure 2.

In Figure 2, β is a value that proves the device's knowledge of π to the server. ρ is a one-time pad by which the server encrypts ν to return it to the device. r is a κ_{sig}-bit value used in the encode function. γ is an encryption of m, r, β and ρ to securely transport them to the server. δ is a message authentication code computed using a, to show the server that this request originated from the device. As in Section 4, δ is not necessary to prove security relative to the goals of Section 3.1, but nevertheless is important in practice to prevent denial-of-service attacks. It is important that the device delete β, d_1, and ρ when the protocol completes, and to never store them on stable storage.

The intuition behind the security of this protocol is similar to that for the GENERIC protocol. The major difference, however, is that only the server's contribution ν to the signature of m is returned to the device, not sk_{dvc} (or the server's share of it). This is what makes key disabling possible, as described in Section 5.1.3.

The efficiency of the S-RSA protocol will generally be worse than the signing efficiency of the underlying RSA signature scheme, not only because of the message and encryption costs, but also because certain optimizations (e.g., Chinese remaindering) that are typically applied for RSA signatures cannot be applied in S-RSA. Nevertheless, since dvc can compute $(\mathsf{encode}(m, r))^{d_1} \bmod N$ while awaiting a response from svr, a significant portion of the device's computation can be parallelized with the server's.

5.1.3 Key disabling

Suppose that the device has been stolen, and that the user wishes to permanently disable the private key of the device. Provided that the user backed up t and τ before the device

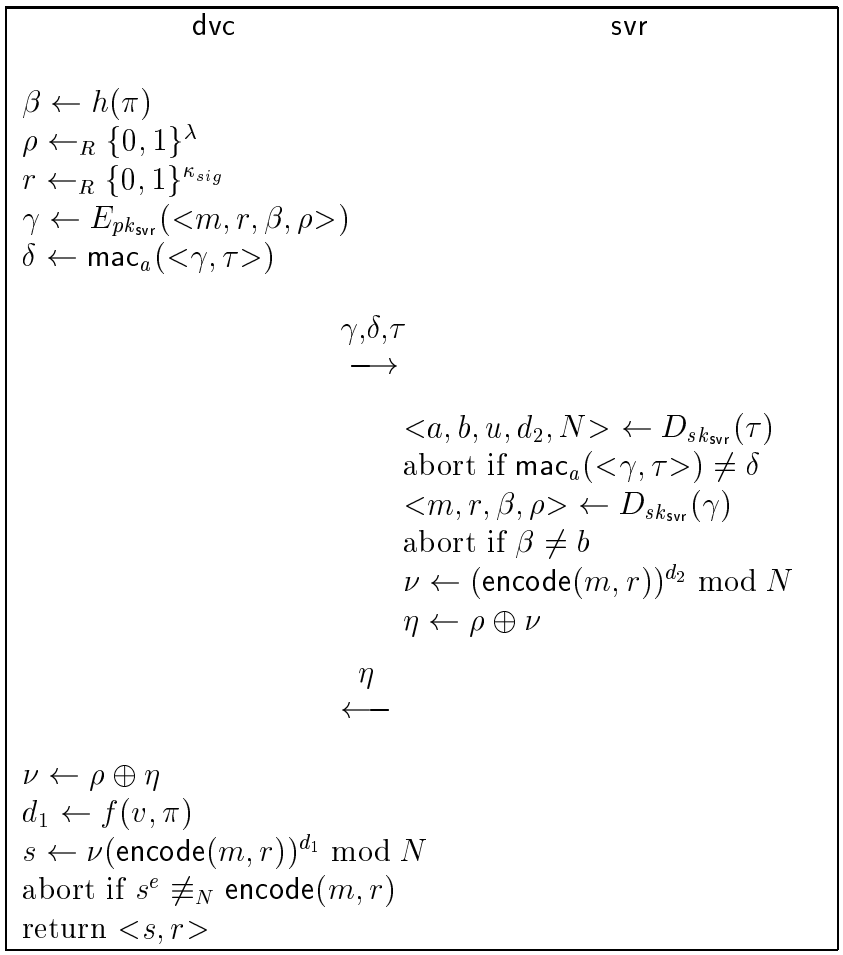

Figure 2: S-RSA signature protocol

was stolen, the user can send t, τ to the server. Upon recovering $<a, b, u, d_2, N> \leftarrow D_{sk_{\mathsf{svr}}}(\tau)$, the server verifies that $u = h_{\mathsf{dsbl}}(t)$ and, if so, records τ on a disabled list. Subsequently, the server should refuse to respond to any request containing the ticket τ. This requires that the server store τ (or a hash of it) on a "blacklist". Rather than storing τ forever, though, the server can discard τ once there is no danger that pk_{dvc} will be used subsequently (e.g., once the public key has been revoked). Note that for security against denial-of-service attacks (an adversary attempting to disable τ without t), we do not need h_{dsbl} to be a random oracle, but simply a one-way hash function.

5.2 D-ElG: a protocol for ElGamal decryption

In this section we give a protocol by which the device can perform decryption with an ElGamal [17] private key, using our techniques to gain the same benefits as S-RSA yielded for RSA signatures. We focus here on decryption (versus signatures), and ElGamal (versus

RSA), to demonstrate the breadth of cryptographic operations to which our techniques apply. Indeed, protocols for decryption with an RSA private key follow naturally from the protocol of Section 5.1. While protocols for signature schemes based on discrete logarithms (e.g., DSA [29]) do not immediately follow from the protocol of this section, they can be achieved using more specialized cryptographic techniques, as corollaries of [33].

For ElGamal encryption, the public and private keys of the device are $pk_{\mathsf{dvc}} = <g, p, q, y>$ and $sk_{\mathsf{dvc}} = <g, p, q, x>$, respectively, where p is an λ-bit prime, g is an element of order q in $\mathbb{Z}_p^*$, x is an element of $\mathbb{Z}_q$ chosen uniformly at random, and $y = g^x \bmod p$. For generality (and reasons that will become clearer later), we describe the D-ElG protocol using an abstract specification of "ElGamal-like" encryption. An *ElGamal-like encryption scheme* is an encryption scheme in which (i) the public and private keys are as above; and (ii) the decryption function D can be expressed in the following form:

$D_{<g,p,q,x>}(c)$: abort if $\mathsf{valid}(c) = 0$
$w \leftarrow \mathsf{select}(c)$
$z \leftarrow w^x \bmod p$
$m \leftarrow \mathsf{reveal}(z, c)$
return m

Above, $\mathsf{valid}(c)$ tests the well-formedness of the ciphertext c; it returns 1 if well-formed and 0 otherwise. $\mathsf{select}(c)$ returns the argument w that is raised to the x-th power modulo p. $\mathsf{reveal}(z, c)$ generates the plaintext m using the result z of that computation. For example, in original ElGamal encryption, where $q = p - 1$ and $c = <c_1, c_2> = <g^k \bmod p, my^k \bmod p>$ for some secret value $k \in \mathbb{Z}_q$, $\mathsf{valid}(<c_1, c_2>)$ returns 1 if $c_1, c_2 \in \mathbb{Z}_p^*$ and 0 otherwise; $\mathsf{select}(<c_1, c_2>)$ returns c_1; and $\mathsf{reveal}(z, <c_1, c_2>)$ returns $c_2 z^{-1} \bmod p$. We note, however, that the private key is *not* an argument to valid, select, or reveal; rather, the private key is used only in computing z. Using this framework, the D-ElG protocol is described in the following subsections. We will discuss various ElGamal-like encryption functions and their use in this protocol in Section 5.2.4.

In the following three sections, we detail the steps of the initialization algorithm, the decryption protocol, and the key disabling protocol. In Section 5.2.4, we describe various instantiations of these protocols using ElGamal-like encryption schemes. In Section 8 this system, instantiated with an ElGamal-like encryption scheme (i.e., an encryption scheme as described above, with a specific definition of security as described in Section 8) is formally proven (in the random oracle model) to meet goals I–IV of Section 3.1.

5.2.1 Device initialization

The inputs to device initialization are the server's public encryption key pk_{svr}, the user's password π_0, the device's public key $pk_{\mathsf{dvc}} = <g, p, q, y>$, and the corresponding private key $sk_{\mathsf{dvc}} = <g, p, q, x>$. The initialization algorithm proceeds as follows:

$t \leftarrow_R \{0,1\}^\kappa$
$u \leftarrow h_{\mathsf{dsbl}}(t)$
$v \leftarrow_R \{0,1\}^\kappa$
$a \leftarrow_R \{0,1\}^\kappa$
$b \leftarrow h(\pi_0)$
$x_1 \leftarrow f(v, \pi_0)$
$x_2 \leftarrow x - x_1 \bmod q$
$\tau \leftarrow E_{pk_{\mathsf{svr}}}(<a, b, u, g, p, q, x_2>)$

Here we assume that f outputs an element of $\{0,1\}^{2|q|}$. The values v, a, τ, pk_{dvc}, pk_{svr} and t are saved on stable storage in the device. All other values, including u, b, x, x_1, x_2, and π_0, are deleted from the device. The values t and τ should be backed up offline for use in disabling if the need arises. The value τ is the device's "ticket" that it uses to access the service.

5.2.2 Decryption protocol

Figure 3 describes the protocol by which the device decrypts a ciphertext c generated using the device's public key in an ElGamal-like encryption scheme. The input provided to the device for this protocol is the input password π, the ciphertext c, and all of the values saved on stable storage in the initialization protocol of Section 5.2.1. In Figure 3, h_{zkp} is assumed to return an element of $\mathbb{Z}_q$.

The reader should observe in Figure 3 that the device's decryption function is implemented jointly by dvc and svr. Moreover, $<\nu, e, s>$ constitutes a noninteractive zero-knowledge proof from svr (the "prover") to dvc (the "verifier") that svr constructed its contribution ν correctly. As before, β is a value that proves the device's knowledge of π to the server. γ is an encryption of c, β, and ρ to securely transport them to the server. δ is a message authentication code computed using a, to show the server that this request originated from the device.

Decryption via the D-ElG protocol is somewhat more costly than decryption in the underlying ElGamal-like encryption scheme. As in S-RSA, we recommend that dvc compute μ while awaiting a response from svr in order to parallelize computation between the two.

5.2.3 Key disabling

Like S-RSA, the D-ElG protocol also supports key disabling. Assuming that the user backed up t and τ before the device was stolen, the user can send t, τ to the server. Upon recovering $<a, b, u, g, p, q, x_2> \leftarrow D_{sk_{\mathsf{svr}}}(\tau)$, the server verifies that $u = h_{\mathsf{dsbl}}(t)$ and, if so, records τ on a disabled list. Subsequently, the server should refuse to respond to any request containing the ticket τ. We remind the reader that this requires the server to store τ (or a hash of it) on a "blacklist". Rather than storing τ forever, though, the server can discard τ once there is no danger that pk_{dvc} will be used subsequently (e.g., once the public key has been revoked).

5.2.4 Choices for ElGamal-like encryption

There are several possibilities for ElGamal-like encryption schemes that, when used to instantiate the description of Figure 3, result in a protocol that provably satisfies goals I–IV. That said, the precise senses in which a particular instance can satisfy goal IV deserve some discussion. The most natural definition of security for key disabling is that an adversary in $\text{ADV}(\{\mathsf{dvc}, \pi_0\})$ who is presented with a ciphertext c *after* the key has been disabled will be unable to decrypt c. A stronger definition for key disabling could require that c remain indecipherable even if c were given to the adversary *before* key disabling occurred, as long as c were not sent to svr before disabling.

If the original ElGamal scheme is secure against indifferent chosen ciphertext attacks [39], then the protocol of Figure 3 can be proven secure in the former sense when instantiated with original ElGamal. However, the security of ElGamal in this sense has not been established, and is an active area of research (e.g., see [35]). There are, however, ElGamal-like encryption schemes that suffice to achieve even the latter, stronger security property, such as the following proposal from [41] called *TDH1*. In this scheme, q is a κ-bit prime factor of $p-1$. Encryption of a message m proceeds as follows:

$E_{<g,p,q,y>}(m)$: $k \leftarrow_R \mathbb{Z}_q$
$c_1 \leftarrow h_1(y^k \bmod p) \oplus m$
$c_2 \leftarrow g^k \bmod p$
$\ell \leftarrow_R \mathbb{Z}_q$
$g' \leftarrow h_2(<c_1, c_2, g^\ell \bmod p>)$
$c_3 \leftarrow (g')^k \bmod p$
$c_4 \leftarrow h_{\mathsf{zkp}}(<g', c_3, (g')^\ell \bmod p>)$
$c_5 \leftarrow \ell + kc_4 \bmod q$

The tuple $<c_1, c_2, c_3, c_4, c_5>$ is the ciphertext. Above, h_1 outputs a value from $\{0,1\}^{|m|}$, and h_2 outputs an element of the subgroup of $\mathbb{Z}_p^*$ generated by g. For example, this can be achieved by defining $h_2(z) = (h'(z))^{(p-1)/q} \bmod p$ for some other hash function h'. Decryption takes the following form:

$\mathsf{valid}(c)$: $<c_1, c_2, c_3, c_4, c_5> \leftarrow c$
$w_1 \leftarrow g^{c_5}(c_2)^{-c_4} \bmod p$
$g' \leftarrow h_2(<c_1, c_2, w_1>)$
$w_2 \leftarrow (g')^{c_5}(c_3)^{-c_4} \bmod p$
return $(c_4 = h_{\mathsf{zkp}}(<g', c_3, w_2>) \wedge (c_2)^q \equiv_p (c_3)^q \equiv_p 1)$

$\mathsf{select}(c)$: $<c_1, c_2, c_3, c_4, c_5> \leftarrow c$
return c_2

$\mathsf{reveal}(z, c)$: $<c_1, c_2, c_3, c_4, c_5> \leftarrow c$
return $h_1(z) \oplus c_1$

A second proposal from [41], called *TDH2*, can also be used to instantiate our protocol and achieve the stronger version of goal IV.

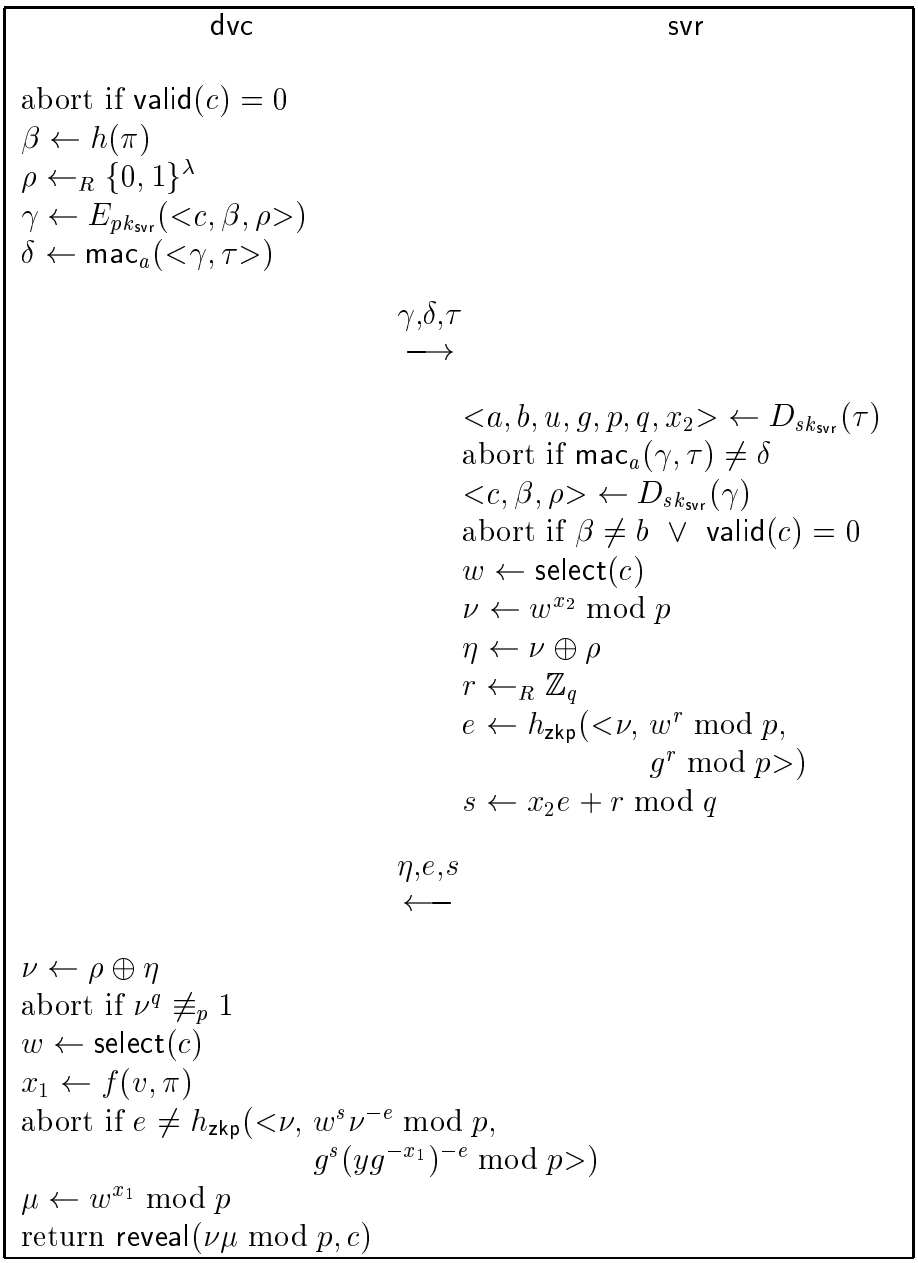

Figure 3: D-ElG decryption protocol

6 Proof of security for GENERIC

In this section we provide a formal proof of security for the GENERIC system in the random oracle model [6]. Hash functions in the random oracle model are assumed to be ideal random functions. That is, for each possible input, each output bit of the hash function is chosen uniformly and independently. Although this is not a standard complexity theoretic assumption [12], it has been useful for proving security for many practical cryptographic schemes (e.g., [7]).

6.1 Definitions

In order to state and prove security of our protocol formally, we must first state requirements for the security of a pseudorandom function, of an encryption scheme, of a signature scheme, and of GENERIC.

Pseudorandom functions A pseudorandom function family is a family of functions f_v parameterized by a secret value v, which has the following security property: It is computationally infeasible to distinguish between an oracle for the f_v function, where v is chosen randomly, and an oracle for a perfectly random function (with the same input and output ranges). See [21] for a formal definition.

Security for encryption schemes We specify adaptive chosen-ciphertext security [39] for an encryption scheme $\mathcal{E} = (G_{enc}, E, D)$. (For more detail, see [3, Property IND-CCA2].) An attacker A is given pk, where $(pk, sk) \leftarrow G_{enc}(1^\lambda)$. A is allowed to query a decryption oracle that takes a ciphertext as input and returns the decryption of that ciphertext (or $\perp$ if the input is not a valid ciphertext). At some point A generates two equal length strings X_0 and X_1 and sends these to a test oracle, which chooses $b \leftarrow_R \{0,1\}$, and returns $Y = E_{pk}(X_b)$. Then A continues as before, with the one restriction that it cannot query the decryption oracle on Y. Finally A outputs b', and succeeds if $b' = b$. We say an attacker A (q, ϵ)-breaks $\mathcal{E}$ if the attacker makes q queries to the decryption oracle, and $2 \cdot \Pr(A \text{ succeeds}) - 1 \geq \epsilon$. Note that this implies

$$\Pr(A \text{ guesses } 0 \mid b = 0) - \Pr(A \text{ guesses } 0 \mid b = 1) \geq \epsilon.$$

Security for signature schemes We specify existential unforgeability versus chosen message attacks [22] for a signature scheme $\mathcal{S} = (G_{sig}, S, V)$. A forger is given pk, where $(pk, sk) \leftarrow G_{sig}(1^\lambda)$, and tries to forge signatures with respect to pk. It is allowed to query a signature oracle (with respect to sk) on messages of its choice. It succeeds if after this it can output a valid forgery (m, σ), where $V_{pk}(m, \sigma) = 1$, but m was not one of the messages signed by the signature oracle. We say a forger (q, ϵ)-breaks a scheme if the forger makes q queries to the signature oracle, and succeeds with probability at least ϵ.

Note: If a scheme uses random oracles, then those oracles are considered part of the scheme, and may be queried by attackers/forgers along with the encryption/signature oracles.

Security for GENERIC Let GENERIC$[\mathcal{S}, \mathcal{E}, \mathcal{D}]$ denote a GENERIC system based on a (matchable) signature scheme $\mathcal{S} = (G_{sig}, S, V)$, an encryption scheme $\mathcal{E}$ for the server, and a dictionary $\mathcal{D}$. A forger is given pk, where $(pk, sk) \leftarrow G_{sig}(1^\lambda)$, and the public data generated by the initialization procedure for the protocol, along with certain secret data of the device and/or server (depending on the type of forger). As in the security definition for standard signature schemes, the goal of the forger is to forge signatures with respect to pk. Instead of a signature oracle, the forger may query a dvc oracle, a svr oracle, and (possibly) random oracles h and f. A random oracle may be queried at any time. It takes an input and returns a random value in the defined range. The svr oracle may be queried at any time by invoking serve with an input message; this either aborts or returns an output message (with respect to the secret server data generated by the initialization procedure).

The dvc oracle may be queried with start, finish, sign, and erase. On a start query, which corresponds to an initiation of the GENERIC protocol, the dvc oracle takes no input, returns an output message, and sets some internal state (with respect to the secret device data and the password generated by the initialization procedure). On a finish query, which corresponds to the device receiving a message ostensibly from the server, the dvc oracle takes an input message and either aborts or returns silently (depending on the message and the internal state set in the previous start, and using the algorithm M from $\mathcal{S}$). On an erase query, the dvc oracle takes no input and returns no output. This is a "placeholder" query that indicates when the user has stopped actively using the device and the device has erased all sensitive information; this is when the device may be vulnerable to capture. On a sign query, the dvc oracle takes a message as input and returns the signature of the message using the algorithm S_{sk} from $\mathcal{S}$. start, finish, sign, and erase must be queried in order, except that sign may be repeated as often as desired, and the complete sequence of four may be repeated as often as desired by the forger. (Of course, each query may have different inputs.) Also, a sign query may only be made between a finish query that does not abort and the next erase query to dvc.

The forger *succeeds* if after this it can output a valid forgery (m, σ), where $V_{pk}(m, \sigma) = 1$, but m was not one of the messages signed by the device with a sign query.

Let q_{sign} be the number of sign queries to dvc. Let q_{dvc} be the number of start queries to dvc. Let q_{svr} be the number of queries to the server that are not the output of a start query. For Theorem 6.2, where we model h and f as random oracles, let q_h and q_f be the number of queries to the respective random oracles. Let q_o be the number of other oracle queries not counted above. Let $\overline{q} = (q_{\mathsf{dvc}}, q_{\mathsf{sign}}, q_{\mathsf{svr}}, q_h, q_f, q_o)$. In a slight abuse of notation, let $|\overline{q}| = q_{\mathsf{dvc}} + q_{\mathsf{sign}} + q_{\mathsf{svr}} + q_h + q_f + q_o$, i.e., the total number of oracle queries.

We say a forger $(\overline{q}, \epsilon)$-breaks GENERIC if it makes $|\overline{q}|$ oracle queries (of the respective type and to the respective oracles) and succeeds with probability at least ϵ. We say GENERIC is $(\overline{q}, \epsilon)$-secure if no forger $(\overline{q}, \epsilon)$-breaks GENERIC.

6.2 Theorems

In this section we prove that if a forger breaks the GENERIC system with probability non-negligibly more than what is inherently possible in a system of this kind (e.g., if the device is stolen, the attacker inherently has the ability to launch an online dictionary attack, but not an offline dictionary attack), then the underlying signature scheme or encryption scheme used in GENERIC can be broken with non-negligible probability. This implies that if the underlying signature and encryption schemes are secure, our system will be as secure as inherently possible.

We prove security separately for the different types of attackers from Section 3.1. The idea behind each proof is a simulation argument. We assume that a forger F can break the GENERIC system, meaning that it can forge a message not signed by the device in the GENERIC system. We then construct a forger F^* that forges in the underlying signature scheme. Basically F^* will run F over a simulation of the GENERIC system, and when F succeeds in forging a signature in GENERIC (in a way not inherently possible, as discussed above), then F^* will succeed in forging a signature in the underlying signature scheme.

In the security proof against a device-compromising (i.e., $\text{ADV}(\{\mathsf{dvc}\})$) forger F, there is a slight complication. If F were able to break the encryption scheme of the server, a forger F^* as described above may not be able to simulate properly. Thus we show that either F forges signatures (in a way not inherently possible) in a simulation where all encryptions are of strings of zeros, and thus we can construct a forger F^* that succeeds in forging a signature in the underlying signature scheme, or F does not forge signatures (in a way not inherently possible) in that simulation, and thus it must be able to distinguish the true encryptions from the zeroed encryptions. Then we can construct an attacker A^* that breaks the underlying encryption scheme. In addition, we note that the device-compromising forgers for which we prove this result are even stronger than allowed in Section 3: after capturing dvc, the forger is permitted to cause dvc to initiate the GENERIC protocol on a message of the forger's choice, with dvc using the correct password π_0 even if the forger does not know π_0. This models a forger that may be able to capture the static data from the device without capturing the device itself, i.e., without the knowledge of the user.

For proving security against all types of forgers, one must assume that both h and f behave as random oracles. However, for certain types of forgers, weaker hash function properties suffice.[2] For proving security against a forger in $\text{ADV}(\{\mathsf{dvc}\})$, we make no requirement on the f function, and we only require h to have a negligible probability of collisions over the dictionary $\mathcal{D}$. If $|\mathcal{D}|$ were polynomial in κ, then it would suffice for h to be a collision resistant hash function. If $|\mathcal{D}|$ were super-polynomial, that property would not suffice. However, it would suffice for h to be a permutation.[3] For proving security against a forger in $\text{ADV}(\{\mathsf{svr}, \pi_0\})$, we make no requirement on the h function, and it would suffice for $\{f_v\}$ to be a pseudorandom function family.

[2]Minimizing reliance on the random oracle model is generally desirable, since random oracles are not a standard cryptographic assumption [12].

[3]Also, certain weaker properties for h would lead to provable security, but with weaker bounds in the theorem.

In the theorems below, we use "$\approx$" to indicate equality to within negligible factors. Moreover, in our simulations, the forger F is run at most once, and so the times of our simulations are straightforward and omitted from our theorem statements.

Theorem 6.1 *Let $\{f_v\}$ be a pseudorandom function family. If a type* $\text{ADV}(\{\mathsf{svr}, \pi_0\})$ *forger $(\overline{q}, \epsilon)$-breaks the* $\text{GENERIC}[\mathcal{S}, \mathcal{E}, \mathcal{D}]$ *system, then there exists a forger F^* that $(q_{\mathsf{sign}}, \epsilon')$-breaks $\mathcal{S}$ with $\epsilon' \approx \epsilon$.*

Proof: Given $F \in \text{ADV}(\{\mathsf{svr}, \pi_0\})$ that $(\overline{q}, \epsilon)$-breaks the $\text{GENERIC}[\mathcal{S}, \mathcal{E}, \mathcal{D}]$ system, we construct a forger F^* for $\mathcal{S}$. F^* is given public key pk from $\mathcal{S}$ and simulates the GENERIC system for F, so that if F constructs a valid forgery, this will also be a valid forgery for $\mathcal{S}$, and thus F^* will succeed.

Simulation: F^* gives pk to F as the device's public signature key. Then F^* generates the server's key pair $(pk_{\mathsf{svr}}, sk_{\mathsf{svr}})$, and gives that to F. Next, F^* generates the secret user password $\pi_0 \leftarrow_R \mathcal{D}$ and gives that to F. Finally, F^* generates the data $<a, b, c>$ for the ticket τ in the normal way, using a random $v \in \{0,1\}^\kappa$, except that c is drawn randomly from $\{0,1\}^\kappa$.

F^* responds to svr queries as in the real protocol. Note that the probability that the adversary guesses b is negligible. F^* also responds to dvc start queries as in the real protocol.

However, for a dvc $\mathsf{finish}(\eta)$ query corresponding to a start query, F^* simply checks that $c = \eta \oplus \rho$ (where ρ was computed in the start query) and has dvc abort if this is false. Then, assuming the dvc did not abort, for any subsequent dvc $\mathsf{sign}(m)$ query before an erase query, F^* calls the signature oracle for $\mathcal{S}$ on m and returns the result.

Analysis: Let GENERIC$'$ be the GENERIC protocol with f_v replaced with a perfectly random function, and let ϵ'' be the probability that F produces a forgery when run against GENERIC$'$. By the pseudorandomness of f, $\epsilon'' \approx \epsilon$. Now let ϵ' be the probability that F forges in the simulation, and hence the probability that F^* forges in the underlying signature scheme. One can see that the simulation above is statistically indistinguishable from GENERIC$'$ to F, and so $\epsilon' \approx \epsilon'' \approx \epsilon$. ∎

Theorem 6.2 *Let h and f be random oracles. If a type* $\text{ADV}(\{\mathsf{dvc}, \mathsf{svr}\})$ *forger $(\overline{q}, \epsilon)$-breaks the* $\text{GENERIC}[\mathcal{S}, \mathcal{E}, \mathcal{D}]$ *system, then there exists a forger F^* that $(q_{\mathsf{sign}}, \epsilon')$-breaks $\mathcal{S}$ with $\epsilon' \approx \epsilon - \frac{q_h + q_f}{|\mathcal{D}|}$.*

Proof: Given $F \in \text{ADV}(\{\mathsf{dvc}, \mathsf{svr}\})$ that $(\overline{q}, \epsilon)$-breaks the $\text{GENERIC}[\mathcal{S}, \mathcal{E}, \mathcal{D}]$ system, we construct a forger F^* for $\mathcal{S}$. F^* is given public key pk from $\mathcal{S}$ and simulates the GENERIC system for F, so that if F constructs a valid forgery, this will also be a valid forgery for $\mathcal{S}$, and thus F^* will succeed.

Simulation: F^* gives pk to F as the device's public signature key. Then F^* generates the server's key pair $(pk_{\mathsf{svr}}, sk_{\mathsf{svr}})$, and gives that to F. Next, F^* generates the secret user password $\pi_0 \leftarrow_R \mathcal{D}$. Finally, F^* generates the data $<a, b, c>$ for the ticket τ in the normal way, using a random $v \in \{0,1\}^\kappa$, except that c is drawn randomly from $\{0,1\}^\kappa$. F^* gives a, v, and $\tau = E_{pk_{\mathsf{svr}}}(<a, b, c>)$ to F.

F^* responds to an $h(\pi)$ or $f(v', \pi)$ query as a normal random oracle would, except that it aborts if $\pi = \pi_0$ (for an $h()$ query) or $\pi = \pi_0$ and $v' = v$ (for an $f()$ query).

F^* responds to svr and dvc queries as in the proof of Theorem 6.1.

Analysis: Unless F makes a query $h(\pi_0)$ or $f(v, \pi_0)$, which occurs with probability at most $\frac{q_h+q_f}{|\mathcal{D}|}$, the simulation is indistinguishable from the real protocol to F, so if F produced a forgery with probability ϵ in GENERIC, F^* would produce a forgery with probability at least $\epsilon' \approx \epsilon - \frac{q_h+q_f}{|\mathcal{D}|}$ in the underlying signature scheme. □

Theorem 6.3 *Suppose h has a negligible probability of collision over $\mathcal{D}$. If a type* ADV($\{$dvc$\}$) *forger $(\overline{q}, \epsilon)$-breaks the* GENERIC$[\mathcal{S}, \mathcal{E}, \mathcal{D}]$ *system where $\epsilon \approx \frac{q_{\mathsf{svr}}}{|\mathcal{D}|} + \psi$, then either there exists an attacker A^* that $(2q_{\mathsf{svr}}, \epsilon')$-breaks $\mathcal{E}$ with $\epsilon' \approx \frac{\psi}{2(1+q_{\mathsf{dvc}})}$, or there exists a forger F^* that $(q_{\mathsf{sign}}, \epsilon'')$-breaks $\mathcal{S}$ with $\epsilon'' \approx \frac{\psi}{2}$.*

Proof: Given $F \in$ ADV($\{$dvc$\}$) that $(\overline{q}, \epsilon)$-breaks the GENERIC$[\mathcal{S}, \mathcal{E}, \mathcal{D}]$ system, we show that either we can construct a forger F^* against $\mathcal{S}$, or an attacker A^* against $\mathcal{E}$. We first show that if forger F *wins* (as defined below) against a certain simulation with probability greater than $\frac{q_{\mathsf{svr}}}{|\mathcal{D}|} + \frac{\psi}{2}$, we can construct a forger F^* that can $(q_{\mathsf{dvc}}, \epsilon'')$-break $\mathcal{S}$ with $\epsilon'' \approx \frac{\psi}{2}$. Assuming F does not win against that simulation with the probability stated above, then we show that we can construct an attacker A^* that breaks $\mathcal{E}$ with probability at least $\frac{\psi}{2(1+q_{\mathsf{dvc}})}$.

Part 1 F^* is given public key pk from $\mathcal{S}$ and simulates the GENERIC system for F, so that if F constructs a valid forgery, this will also be a valid forgery for $\mathcal{S}$, and thus F^* will succeed. We say F *wins* against this simulation if F produces a valid forgery in the simulation, or if F *makes a successful online password guess.* This is defined as F making a server query with input (γ, δ, τ'), where $\delta = \mathsf{mac}_a(<\gamma, \tau>)$ for the mac key a stored on the device, and either (1) τ' is the ticket stored on the device and γ is a ciphertext not generated by a device start query, and where $<\beta, \rho> \leftarrow D_{sk_{\mathsf{svr}}}(\gamma)$, and $\beta = h(\pi_0)$; or (2) τ' is not the ticket stored on the device but γ was generated by a device start query, and where $<a', b', c'> \leftarrow D_{sk_{\mathsf{svr}}}(\tau')$, and $b' = h(\pi_0)$.

Part 1 Simulation: F^* gives pk to F as the device's public signature key. Then F^* generates the server's key pair $(pk_{\mathsf{svr}}, sk_{\mathsf{svr}})$, and gives pk_{svr} to F. Next, F^* generates the secret user password $\pi_0 \leftarrow_R \mathcal{D}$. Finally F^* generates the data $<a, b, c>$ for the ticket τ in the normal way, using a random $v \in \{0,1\}^\kappa$, except that c is drawn randomly from $\{0,1\}^\kappa$. F^* gives a, v, and $\tau = E_{pk_{\mathsf{svr}}}(<0^{2\kappa+\lambda}>)$ to F.

F^* responds to svr serve(γ, δ, τ') queries as follows:

Case 1: (γ, δ, τ') is from a dvc start query: Return $\rho \oplus c$ (where ρ was from the start query).

Case 2: γ and τ' are from a dvc start query, but not δ: Have svr abort.

Case 3: $\tau' = \tau$, but γ is not from a dvc start query: Verify δ like a normal server, but using the a value from initialization as the mac key. Then compute $<\beta, \rho> \leftarrow D_{sk_{\mathsf{svr}}}(\gamma)$. Abort the simulation if $\beta = b$ (this is a successful online password guess), and have svr abort if $\beta \neq b$.

Case 4: $\tau' \neq \tau$, but γ is from a dvc start query: Compute $<a', b', c'> \leftarrow D_{sk_{\mathsf{svr}}}(\tau')$. Verify δ like a normal server, using mac key a'. Abort the simulation if $b' = b$ (this is a successful

online password guess), and have svr abort if $b' \neq b$.
Case 5: $\tau' \neq \tau$ and γ is not from a dvc start query: Behave like a normal server.

F^* responds to a dvc start query as a normal dvc would, except setting $\gamma \leftarrow E_{pk_{\mathsf{svr}}}(0^{\kappa+\lambda})$. For a dvc finish($\eta$) query corresponding to a start query, F^* simply checks that $c = \eta \oplus \rho$ (where ρ was computed in the start query) and has the dvc abort if this is false. Then, assuming the dvc did not abort, for any subsequent dvc sign(m) query before an erase query, F^* calls the signature oracle for $\mathcal{S}$ on m and returns the result.

Part 1 Analysis: The probability that F makes a successful online password guess is at most $\frac{q_{\mathsf{svr}}}{|\mathcal{D}|}$, disregarding negligible probabilities (since π_0 was chosen randomly and h has a negligible probability of collision over $\mathcal{D}$), so if F wins against the simulation with probability at least $\frac{q_{\mathsf{svr}}}{|\mathcal{D}|} + \frac{\psi}{2}$, it produces a forgery with probability at least $\frac{\psi}{2}$, and thus F^* produces a forgery in the underlying signature scheme with probability at least $\epsilon'' \approx \frac{\psi}{2}$.

Part 2 For the second part of the proof, we assume that the probability of F winning in Part 1 is at most $\frac{q_{\mathsf{svr}}}{|\mathcal{D}|} + \frac{\psi}{2}$. Then we construct an attacker A^* that breaks $\mathcal{E}$ with probability $\frac{\psi}{2(1+q_{\mathsf{dvc}})}$. Our attacker A^* is given a public key pk' from $\mathcal{E}$, and runs a simulation of the GENERIC system for F.

First consider a simulator that gives pk' to F as the server's public encryption key, and then simulates GENERIC exactly, but using a decryption oracle to decrypt messages encrypted under key pk' by F. Note that the decryptions of τ and any γ generated by the dvc would already be known to the simulator. Now consider the same simulation, but with the normal messages encrypted by the simulator replaced with strings of zeros. (Naturally, the server pretends the encryptions are of the normal messages, not the strings of zeros.) The latter simulation is equivalent to the Part 1 simulation, except that the simulator does not abort on a successful online password guess. Still, the probability of F forging in the latter simulation is at most $\frac{q_{\mathsf{svr}}}{|\mathcal{D}|} + \frac{\psi}{2}$, while the probability of F forging in the former simulation is at least $\frac{q_{\mathsf{svr}}}{|\mathcal{D}|} + \psi$.

Now we use a standard hybrid argument to construct A^*. Let experiment $j \in \{0, \ldots, q_{\mathsf{dvc}}+1\}$ correspond to the first j ciphertexts (generated by the simulator) be of the normal messages, and the remainder be encryptions of strings of 0's, and let p_j be the probability of F forging in experiment j. Then the average value for $i \in \{0, \ldots, q_{\mathsf{dvc}}\}$ of $p_{i+1} - p_i$ is at least $\frac{\psi}{2(1+q_{\mathsf{dvc}})}$.

Therefore, to construct A^*, we simply have A^* choose a random value $i \in \{0, \ldots, q_{\mathsf{dvc}}\}$, and run experiment i as above, but calling 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 A^* outputs 0 if F forges (meaning it believes X_0 was encrypted by the test oracle), and 1 otherwise. By the analysis above, A^* breaks $\mathcal{E}$ with probability $\frac{\psi}{2(1+q_{\mathsf{dvc}})}$. ∎

7 Proof of security for S-RSA

In this section we provide a formal proof of security for the S-RSA system in the random oracle model.

7.1 Definitions

In order to state and prove security of our protocol formally, we must first state requirements for the security of S-RSA.

Security for S-RSA Let S-RSA$[\mathcal{E}, \mathcal{D}]$ denote an S-RSA system based on an encryption scheme $\mathcal{E}$ for the server and dictionary $\mathcal{D}$. A forger is given $<e, N>$ where

$$(<e, N>, <d, N, \phi(N)>) \leftarrow G_{RSA}(1^{\lambda}),$$

the public data generated by the initialization procedure for the protocol, and certain secret data of the device, server, and/or the user's password (depending on the type of forger). The goal of the forger is to forge RSA signatures with respect to $<e, N>$. There is a dvc oracle, a disable oracle, a svr oracle, and (possibly) random oracles h and f. A random oracle may be queried at any time. It takes an input and returns a random hash of that input, in the defined range. The disable oracle may be queried with getVals. It responds with a value t and the device's ticket τ.

The svr oracle may be queried with serve and disable. On a serve(γ, δ, τ) query, which represents the receipt of a message in the S-RSA protocol ostensibly from the device, it either aborts or returns an output message η (with respect to the secret server data generated by the initialization procedure). On a disable(t, τ) query, which represents a disable request, the svr oracle rejects all future queries with the ticket τ if t corresponds to τ (see Section 5.1.3).

The dvc oracle may be queried with start and finish. We assume there is an implicit notion of sessions so that the dvc oracle can determine the start query corresponding to a finish query. On a start(m) query, which represents a request to initiate the S-RSA protocol, the dvc returns an output message $<\gamma, \delta, \tau>$, and sets some internal state (with respect to the secret device data and the password generated by the initialization procedure). On the corresponding finish(η) query, which represents the device's receipt of a response ostensibly from the server, the dvc oracle either aborts or returns a valid signature for the message m given as input to the previous start query.

A forger of type Adv$(\{\mathsf{svr}, \pi_0\})$, Adv$(\{\mathsf{dvc}, \mathsf{svr}\})$, or Adv$(\{\mathsf{dvc}\})$ *succeeds* if after this it can output a pair $(m, <s, r>)$ where $s^e \equiv_N$ encode(m, r) and there was no start(m) query. A type Adv$(\{\mathsf{dvc}, \pi_0\})$ forger *succeeds* if after this it can output a pair $(m, <s, r>)$ where $s^e \equiv_N$ encode(m, r) and there was no serve(γ, δ, τ) query, where $D_{sk_{\mathsf{svr}}}(\gamma) = <m, *, *>$, before a disable$(t, \tau)$ query that disables the device's ticket τ.

Let q_{dvc} be the number of start queries to the device. Let q_{svr} be the number of server serve queries. For Theorem 7.2, where we model h and f as random oracles, let q_h and q_f be the number of queries to the respective random oracles. Let q_o be the number of other

oracle queries not counted above. Let $\overline{q} = (q_{\mathsf{dvc}}, q_{\mathsf{svr}}, q_o, q_h, q_f)$. In a slight abuse of notation, let $|\overline{q}| = q_{\mathsf{dvc}} + q_{\mathsf{svr}} + q_o + q_h + q_f$, i.e., the total number of oracle queries. We say a forger $(\overline{q}, \epsilon)$-breaks S-RSA if it makes $|\overline{q}|$ oracle queries (of the respective type and to the respective oracles) and succeeds with probability at least ϵ.

7.2 Theorems

Here we prove that if a forger breaks the S-RSA system with probability non-negligibly more than what is inherently possible in a system of this kind then either the underlying RSA signature scheme or the underlying encryption scheme used in S-RSA can be broken with non-negligible probability. This implies that if the underlying RSA signature scheme and the underlying encryption scheme are secure, our system will be as secure as inherently possible. As in Section 6, we prove security separately for the different types of attackers from Section 3.1. One difference is that here we also prove security for a forger in $\textsc{Adv}(\{\mathsf{dvc}, \pi_0\})$, and in this case we make no requirement on either h or f.

Theorem 7.1 *Let $\{f_v\}$ be a pseudorandom function family. If a type $\textsc{Adv}(\{\mathsf{svr}, \pi_0\})$ forger $(\overline{q}, \epsilon)$-breaks the S-RSA$[\mathcal{E}, \mathcal{D}]$ system, then there exists a forger that $(q_{\mathsf{dvc}}, \epsilon')$-breaks the underlying RSA signature scheme with $\epsilon' \approx \epsilon$.*

Proof: Given $F \in \textsc{Adv}(\{\mathsf{svr}, \pi_0\})$ that $(\overline{q}, \epsilon)$-breaks the S-RSA$[\mathcal{E}, \mathcal{D}]$ system, we construct a forger F^* for the underlying RSA signature scheme. F^* is given public key $<e, N>$ for the RSA signature scheme and simulates the S-RSA system for F, so that any forgery constructed by F will be a forgery in the underlying RSA signature scheme.

Simulation: F^* gives $<e, N>$ to F as the device's public signature key. Then F^* generates the server's key pair $(pk_{\mathsf{svr}}, sk_{\mathsf{svr}})$, and gives that to F. Next F^* generates the user password $\pi_0 \leftarrow_R \mathcal{D}$ and gives that to F. Finally F^* generates $<a, b, u, d_2, N>$ for the ticket τ in the normal way, using random $t, v \in \{0,1\}^\kappa$, except that d_2 is drawn randomly from $\mathbb{Z}_N$.

F^* responds to svr queries as in the real protocol. F^* responds to $\mathsf{dvc}\ \mathsf{start}(m)$ queries by querying the signature oracle to get $\sigma = <s, r>$ and then responding as in the real protocol using that r value. F^* responds to a $\mathsf{dvc}\ \mathsf{finish}(\eta)$ query corresponding to a $\mathsf{start}(m)$ query by computing $\nu = \eta \oplus \rho$ (where ρ was computed in the $\mathsf{start}(m)$ query) and checking that $\nu \equiv_N (\mathsf{encode}(m, r))^{d_2}$. If this is false, F^* has dvc abort. Otherwise, F^* returns the σ returned from the signature oracle query in the start query. F^* responds to a $\mathsf{getVals}$ query to the disable oracle by returning t, τ.

Analysis: Let S-RSA$'$ be the S-RSA protocol with f_v replaced with a perfectly random function, and let ϵ'' be the probability that F produces a forgery when run against S-RSA$'$. By the pseudorandomness of f, $\epsilon'' \approx \epsilon$. Now let ϵ' be the probability that F forges in the simulation, and hence the probability that F^* forges in the underlying RSA signature scheme. One can see that the simulation above is statistically indistinguishable from S-RSA$'$ to F, and so $\epsilon' \approx \epsilon'' \approx \epsilon$. ∎

Theorem 7.2 *Let h and f be random oracles. If a type* $\text{ADV}(\{\mathsf{dvc}, \mathsf{svr}\})$ *forger $(\overline{q}, \epsilon)$-breaks the* S-RSA$[\mathcal{E}, \mathcal{D}]$ *system, then there exists a forger F^* that $(q_{\mathsf{dvc}}, \epsilon')$-breaks the underlying RSA signature scheme with $\epsilon' \approx \epsilon - \frac{q_h + q_f}{|\mathcal{D}|}$.*

Proof: Given $F \in \text{ADV}(\{\mathsf{dvc}, \mathsf{svr}\})$ that $(\overline{q}, \epsilon)$-breaks the S-RSA$[\mathcal{E}, \mathcal{D}]$ system, we construct a forger F^* for the underlying RSA signature scheme. F^* is given public key $<e, N>$ for the RSA signature scheme and simulates the S-RSA system for F, so that any forgery construct by F without F guessing the password (as described below) will be a forgery in the underlying RSA signature scheme.

Simulation: F^* gives $<e, N>$ to F as the device's public signature key. Then F^* generates the server's key pair $(pk_{\mathsf{svr}}, sk_{\mathsf{svr}})$, and gives that to F. Next F^* generates the secret user password $\pi_0 \leftarrow_R \mathcal{D}$. Finally, F^* generates the data $<a, b, u, d_2, N>$ for the ticket τ in the normal way, using random $t, v \in \{0,1\}^\kappa$, except that d_2 is drawn randomly from $\mathbb{Z}_N$. F^* gives a, v, and $\tau = E_{pk_{\mathsf{svr}}}(<a, b, u, d_2, N>)$ to F.

F^* responds to an $h(\pi)$ or $f(v', \pi)$ query as a normal random oracle would, except that it aborts if $\pi = \pi_0$ (for an $h()$ query) or $\pi = \pi_0$ and $v' = v$ (for an $f()$ query).

A responds to queries to the svr, dvc, and disable oracles as in the proof of Theorem 7.1.

Analysis: Unless F makes a query $h(\pi_0)$ or $f(v, \pi_0)$, which occurs with probability at most $\frac{q_h + q_f}{|\mathcal{D}|}$, the simulation is indistinguishable from the real protocol to F, so if F produced a forgery with probability ϵ in S-RSA, F^* would produce a forgery with probability at least $\epsilon' \approx \epsilon - \frac{q_h + q_f}{|\mathcal{D}|}$ in the underlying RSA signature scheme. □

Theorem 7.3 *Suppose h has a negligible probability of collision over $\mathcal{D}$. If a type* $\text{ADV}(\{\mathsf{dvc}\})$ *forger $(\overline{q}, \epsilon)$-breaks the* S-RSA$[\mathcal{E}, \mathcal{D}]$ *system where $\epsilon = \frac{q_{\mathsf{svr}}}{|\mathcal{D}|} + \psi$, then either there exists an attacker A^* that $(2q_{\mathsf{svr}}, \epsilon')$-breaks $\mathcal{E}$ with $\epsilon' \approx \frac{\psi}{2(1+q_{\mathsf{dvc}})}$, or there exists a forger F^* that $(q_{\mathsf{dvc}}, \epsilon'')$-breaks the underlying RSA signature scheme with $\epsilon'' \approx \frac{\psi}{2}$.*

Proof: Given $F \in \text{ADV}(\{\mathsf{dvc}\})$ that $(\overline{q}, \epsilon)$-breaks the S-RSA$[\mathcal{E}, \mathcal{D}]$ system, we show that either we can construct a forger F^* for the underlying RSA signature scheme, or an attacker A^* against $\mathcal{E}$. We first show that if forger F *wins* (as defined below) against a certain simulation with probability greater than $\frac{q_{\mathsf{svr}}}{|\mathcal{D}|} + \frac{\psi}{2}$, we can construct a forger F^* that can $(q_{\mathsf{dvc}}, \epsilon'')$-break the underlying RSA signature scheme with $\epsilon'' \approx \frac{\psi}{2}$. Assuming F does not win against that simulation with the probability stated above, then we show that we can construct an attacker A^* that $(2q_{\mathsf{svr}}, \epsilon')$-breaks $\mathcal{E}$ with $\epsilon' \approx \frac{\psi}{2(1+q_{\mathsf{dvc}})}$.

Part 1 F^* is given public key $<e, N>$ for the RSA signature scheme and simulates the S-RSA system for F. We say F *wins* against the simulation if F produces a valid forgery or if F *makes a successful online password guess.* This is defined as F making a server query with input (γ, δ, τ'), where $\delta = \mathsf{mac}_a(<\gamma, \tau>)$ for the mac key a stored on the device, and either (1) τ' is the ticket stored on the device and γ is a ciphertext not generated by a device start query, and where $<m, r, \beta, \rho> \leftarrow D_{sk_{\mathsf{svr}}}(\gamma)$, and $\beta = h(\pi_0)$; or (2) τ' is not the ticket stored on the device but γ was generated by a device start query, and where $<a', b', u', d_2', N'> \leftarrow D_{sk_{\mathsf{svr}}}(\tau')$, and $b' = h(\pi_0)$.

Part 1 Simulation: F^* gives $<e, N>$ to F as the device's public signature key. Then F^* generates the server's key pair $(pk_{\mathsf{svr}}, sk_{\mathsf{svr}})$, and gives pk_{svr} to F. Next F^* generates the secret user password $\pi_0 \leftarrow_R \mathcal{D}$. Finally, F^* generates $<a, b, u, d_2, N>$ for the ticket τ in the normal way, using random $t, v \in \{0,1\}^\kappa$, except that d_2 is drawn randomly from $\mathbb{Z}_N$. F^* gives a, v, and $\tau = E_{pk_{\mathsf{svr}}}(0^{3\kappa+2\lambda})$ to F.

F^* responds to a $\mathsf{getVals}$ query to the disable oracle by returning t, τ.

F^* responds to a $\mathsf{svr\ disable}(t', \tau')$ query as a normal server would, but using the u value generated in the initialization if $\tau' = \tau$. F^* responds to $\mathsf{svr\ serve}(\gamma, \delta, \tau')$ queries for a τ' that has not been disabled as follows:

Case 1: (γ, δ, τ') is from a $\mathsf{dvc\ start}(m)$ query: Return $\rho \oplus ((\mathsf{encode}(m, r))^{d_2} \bmod N)$ where m, r, and ρ were from the start query.

Case 2: γ and τ' are from a $\mathsf{dvc\ start}$ query, but not δ: Have svr abort.

Case 3: $\tau' = \tau$, but γ is not from a $\mathsf{dvc\ start}$ query: Verify the mac like a normal server, but using the a value from initialization as the mac key. Then compute $<m, r, \beta, \rho> \leftarrow D_{sk_{\mathsf{svr}}}(\gamma)$. Abort the simulation if $\beta = b$ (this is a successful online password guess), and have svr abort if $\beta \neq b$.

Case 4: $\tau' \neq \tau$, but γ is from a $\mathsf{dvc\ start}$ query: Compute $<a', b', u', d_2', N'> \leftarrow D_{sk_{\mathsf{svr}}}(\tau')$. Verify the mac like a normal server, using mac key a'. Abort the simulation if $b' = b$ (this is a successful online password guess), and have svr abort if $b' \neq b$.

Case 5: $\tau' \neq \tau$ and γ is not from a $\mathsf{dvc\ start}$ query: Behave like a normal server.

F^* responds to a $\mathsf{dvc\ start}(m)$ query as in Theorem 7.1, except setting $\gamma \leftarrow E_{pk_{\mathsf{svr}}}(0^{|m|+\kappa_{sig}+\kappa+\lambda})$. F^* responds to a $\mathsf{dvc\ finish}(\eta)$ query as in Theorem 7.1.

Part 1 Analysis: The probability that F makes a successful online password guess is at most $\frac{q_{\mathsf{svr}}}{|\mathcal{D}|}$, disregarding negligible probabilities (since π_0 was chosen randomly and h has a negligible probability of collision over $\mathcal{D}$), so if F wins against the simulation with probability at least $\frac{\psi}{2} + \frac{q_{\mathsf{svr}}}{|\mathcal{D}|}$, it produces a forgery with probability at least $\frac{\psi}{2}$, and thus F^* produces a forgery in the underlying RSA signature scheme with probability at least $\epsilon'' \approx \frac{\psi}{2}$.

Part 2 For the second part of the proof, we assume that the probability of F winning in Part 1 is at most $\frac{q_{\mathsf{svr}}}{|\mathcal{D}|} + \frac{\psi}{2}$. Then we construct an attacker A^* that breaks $\mathcal{E}$ with probability $\frac{\psi}{2(1+q_{\mathsf{dvc}})}$. Our attacker A^* is given a public key pk' from $\mathcal{E}$, and runs a simulation of the S-RSA system for F.

First consider a simulator that gives pk' to F as the server's public encryption key, and then simulates S-RSA exactly, but using a decryption oracle to decrypt messages encrypted under key pk' by the adversary. There will be at most $2q_{\mathsf{svr}}$ of these. (Note that the decryptions of τ and any γ generated by the dvc would already be known to the simulator.) This simulation would be perfectly indistinguishable from the real protocol to F. Now consider the same simulation, but with the ticket and all γ values generated by the device changed to encryptions of strings of zeros. (Naturally, the server pretends the encryptions are of the normal messages, not strings of zeros.) The latter simulation is equivalent to the Part 1 simulation, except that the attacker does not abort on a successful online password guess. Still, the probability of F forging in the latter simulation is at most $\frac{q_{\mathsf{svr}}}{|\mathcal{D}|} + \frac{\psi}{2}$, while

the probability of F forging in the former simulation is at least $\frac{q_{\mathsf{svr}}}{|\mathcal{D}|} + \psi$.

Now we use a standard hybrid argument to construct A^*. Let experiment $j \in \{0, \ldots, q_{\mathsf{dvc}}+1\}$ correspond to the first j ciphertexts (generated by A^*) be of the normal messages, and the remainder be encryptions of strings of 0's, and let p_j be the probability of F forging in experiment j. Then the average value for $i \in \{0, \ldots, q_{\mathsf{dvc}}\}$ of $p_{i+1} - p_i$ is at least $\frac{\psi}{2(1+q_{\mathsf{dvc}})}$.

Therefore, to construct A^*, we simply have A^* choose a random value $i \in \{0, \ldots, q_{\mathsf{dvc}}\}$, and run experiment i as above, but calling 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 A^* outputs 0 if F forges (meaning it believes X_0 was encrypted by the test oracle), and 1 otherwise. By the analysis above, A^* breaks $\mathcal{E}$ with probability $\frac{\psi}{2(1+q_{\mathsf{dvc}})}$, disregarding negligible probabilities. □

Theorem 7.4 *Suppose the underlying RSA signature scheme is deterministic (i.e., $\kappa_{sig} = 0$). If a type* $\textsc{Adv}(\{\mathsf{dvc}, \pi_0\})$ *forger $(\overline{q}, \epsilon)$-breaks the* S-RSA$[\mathcal{E}, \mathcal{D}]$ *system, then either there exists an attacker A^* that $(2q_{\mathsf{svr}}, \epsilon')$-breaks $\mathcal{E}$ with $\epsilon' \approx \frac{\epsilon}{2(1+q_{\mathsf{dvc}})}$, or there exists a forger F^* that $(2q_{\mathsf{svr}}, \epsilon'')$-breaks the underlying RSA signature scheme with $\epsilon'' \approx \frac{\epsilon}{2}$.*

Proof: Given $F \in \textsc{Adv}(\{\mathsf{dvc}, \pi_0\})$ that $(\overline{q}, \epsilon)$-breaks the S-RSA$[\mathcal{E}, \mathcal{D}]$ system, we show that either we can construct a forger F^* for the underlying RSA signature scheme, or an attacker A^* against $\mathcal{E}$. We first construct a forger F^* that runs F against a simulation of S-RSA, such that if F forges in the simulation with probability $\frac{\epsilon}{2}$, F^* $(2q_{\mathsf{svr}}, \epsilon'')$-breaks the underlying RSA signature scheme with $\epsilon'' \approx \frac{\epsilon}{2}$. Assuming F does not forge a signature in that simulation with probability $\frac{\epsilon}{2}$ then we show that we can construct an attacker A^* that $(2q_{\mathsf{svr}}, \epsilon')$-breaks $\mathcal{E}$ with $\epsilon' \approx \frac{\epsilon}{2(1+q_{\mathsf{dvc}})}$

Part 1 F^* is given public key $<e, N>$ for the RSA signature scheme and simulates the S-RSA system for F.

Part 1 Simulation: F^* gives $<e, N>$ to F as the device's public signature key. Then F^* generates the server's key pair $(pk_{\mathsf{svr}}, sk_{\mathsf{svr}})$, and gives pk_{svr} to F. Next F^* generates $\pi_0 \leftarrow_R \mathcal{D}$ and gives π_0 to F. Finally, F^* generates the data a, b, u, and d_1 in the normal way, using random $t, v \in \{0,1\}^\kappa$, but computes $\tau = E_{pk_{\mathsf{svr}}}(0^{3\kappa+2\lambda})$. F^* gives a, v, and τ to F.

F^* responds to a disable oracle query as in the real system. F^* responds to a svr $\mathsf{disable}(t', \tau')$ query as a normal server would, but using the u value generated in the initialization if $\tau' = \tau$.

F^* responds to svr $\mathsf{serve}(\gamma, \delta, \tau')$ queries for a τ' that has not been disabled as follows:
Case 1: (γ, δ, τ') is from a dvc $\mathsf{start}(m)$ query: Query the signature oracle to get $<s, r>$ (where $|r| = 0$, since the signature scheme is deterministic), and then return

$$\rho \oplus (s/(\mathsf{encode}(m, r))^{d_1} \bmod N),$$

where ρ is from the start query.
Case 2: γ and τ' are from a dvc start query, but not δ: Have svr abort.
Case 3: $\tau' = \tau$, but γ is not from a dvc start query: Verify δ like a normal server, but

using the a value from initialization as the mac key. Then compute $<m, r, \beta, \rho> \leftarrow D_{sk_{\mathsf{svr}}}(\gamma)$. Using the b value from initialization, if $\beta = b$, have svr abort. Otherwise, query the signature oracle to get $\sigma = <s, r>$, and return $\rho \oplus (s/(\mathsf{encode}(m, r))^{d_1} \bmod N)$.
Case 4: $\tau' \neq \tau$, but γ is from a dvc start(m) query: Say $<a', b', u', d_2', N'> \leftarrow D_{sk_{\mathsf{svr}}}(\tau')$. Behave like a normal server, but using m, r, β, and ρ from the dvc start(m) query.
Case 5: $\tau' \neq \tau$ and γ is not from a dvc start query: Behave like a normal server.

F^* responds to a dvc start(m) query as a normal dvc would, except setting $\gamma \leftarrow E_{pk_{\mathsf{svr}}}(0^{|m|+\kappa+\lambda})$. For a dvc finish$(\eta)$ query corresponding to a start(m) query that returned (γ, δ, τ), if η was not returned from a svr serve(γ, δ, τ) query, have dvc abort. Otherwise F^* returns the signature found in that serve(γ, δ, τ) query.

Part 1 Analysis: If F forges in the simulation, then F^* forges in the underlying signature scheme.

Part 2 The second part of the proof is similar to Part 2 of the proof of Theorem 7.3, except with ψ replaced by ϵ, and no $\frac{q_{\mathsf{svr}}}{|\mathcal{D}|}$ term. □

8 Proof of security for D-ElG

In this section we provide a proof of security for the D-ElG system instantiated with an ElGamal-like encryption scheme.

8.1 Definitions

We first define security for an ElGamal-like (ElGL) encryption scheme, and for the D-ElG protocol itself.

Security for ElGL encryption schemes The security for an ElGL encryption scheme is defined exactly like the security for a standard encryption scheme, except for the definition of the decryption oracle. Assuming that the public/secret key pair is

$$(<g, p, q, y>, <g, p, q, x>) \leftarrow G_{\mathsf{ElGL}}(1^\lambda),$$

when the decryption oracle receives a ciphertext c, it does not return the decryption of c, but simply $z = (\mathsf{select}(c))^x \bmod p$, from which the decryption of c can be computed using reveal(z, c).

Note: The TDH1 and TDH2 encryption schemes from [41], when restricted to a single server, are in fact ElGL encryption schemes.

Security for D-ElG Let D-ElG$[\mathcal{E}, \mathcal{D}]$ denote a D-ElG system based on an encryption scheme $\mathcal{E}$ for the server and dictionary $\mathcal{D}$. An attacker A is given $<g, p, q, y>$ where

$$(<g, p, q, y>, <g, p, q, x>) \leftarrow G_{\mathsf{ElGL}}(1^\lambda),$$

the public data generated by the initialization procedure for the protocol, and certain secret data of the device, server, and/or the user's password (depending on the type of forger). There is a test oracle, dvc oracle, a disable oracle, a svr oracle, random oracle h_{zkp}, and (possibly) random oracles h and f. A may query the disable oracle and the random oracles as in S-RSA. Queries to the dvc and svr oracles are also similar to S-RSA, with the following changes. A may query the dvc oracle with a $\mathsf{start}(c)$ query, which corresponds to a protocol initiation on ElGL ciphertext c, and a $\mathsf{finish}(\eta, e, s)$ query. A may query the svr oracle with a $\mathsf{serve}(\gamma, \delta, \tau)$ query corresponding to an invocation ostensibly by the device, which returns a tuple η, e, s. At some point A generates two equal length strings X_0 and X_1 and sends these to the test oracle, which chooses $b \leftarrow_R \{0,1\}$, and returns the ElGL encryption $Y = E_{pk}(X_b)$. Then A continues as before, with the restriction that if A is of type $\textsc{Adv}(\{\mathsf{svr}, \pi_0\})$, $\textsc{Adv}(\{\mathsf{dvc}, \mathsf{svr}\})$, or $\textsc{Adv}(\{\mathsf{dvc}\})$, then it cannot query the dvc with a $\mathsf{start}(Y)$ query, and if A is of type $\textsc{Adv}(\{\mathsf{dvc}, \pi_0\})$, then it cannot query the svr with $\mathsf{serve}(\gamma, \delta, \tau)$ query, where $D_{sk_{\mathsf{svr}}}(\gamma) = <Y, *, *>$, before a $\mathsf{disable}(t, \tau)$ query that disables the device's ticket τ. Finally A outputs b', and succeeds if $b' = b$.

Let q_{dvc}, q_{svr}, q_h, q_f, q_o, $\overline{q}$, and $|\overline{q}|$ be defined in the same manner as in Section 7. We say that A $(\overline{q}, \epsilon)$-breaks D-ElG if A makes $|\overline{q}|$ oracle queries (of the respective types and to the respective oracles), and $2 \cdot \Pr(A \text{ succeeds}) - 1 \geq \epsilon$. Note that this implies

$$\Pr(A \text{ guesses } 0 \mid b = 0) - \Pr(A \text{ guesses } 0 \mid b = 1) \geq \epsilon.$$

8.2 Theorems

Here we prove that if a forger breaks the D-ElG system with probability non-negligibly more than what is inherently possible in a system of this kind then either the underlying ElGL encryption scheme or the underlying server encryption scheme used in D-ElG can be broken with non-negligible probability. This implies that if the underlying ElGL encryption scheme and the underlying server encryption scheme are secure, our system will be as secure as inherently possible. As in Section 7, we prove security separately for the different types of attackers from Section 3.1.

Theorem 8.1 *Let $\{f_v\}$ be a pseudorandom function family. If a type $\textsc{Adv}(\{\mathsf{svr}, \pi_0\})$ attacker $(\overline{q}, \epsilon)$-breaks the D-ElG$[\mathcal{E}, \mathcal{D}]$ system, then there exists an attacker A^* that $(q_{\mathsf{dvc}}, \epsilon')$-breaks the underlying ElGL scheme with $\epsilon' \approx \epsilon$.*

Proof: Given $A \in \textsc{Adv}(\{\mathsf{svr}, \pi_0\})$ that $(\overline{q}, \epsilon)$-breaks the D-ElG$[\mathcal{E}, \mathcal{D}]$ system, we construct an attacker A^* for the underlying ElGL scheme. A^* is given public key $<g, p, q, y>$ for the ElGL scheme and simulates the D-ElG system for A, such that if A succeeds in attacking D-ElG, A^* will succeed in attacking the underlying ElGL scheme.

Simulation: A^* gives $<g, p, q, y>$ to A as the device's public encryption key. Then A^* generates the server's key pair $(pk_{\mathsf{svr}}, sk_{\mathsf{svr}})$, and gives that to A. Next A^* generates the data $<a, b, u, g, p, q, x_2>$ for the ticket τ in the normal way, using random $t, v \in \{0,1\}^\kappa$, except that x_2 is drawn randomly from $\mathbb{Z}_q$.

A^* responds to svr and disable queries as in the real protocol. A^* also responds to dvc $\mathsf{start}(c)$ queries as in the real protocol. A^* responds to queries to the test oracle by forwarding the query to the underlying ElGL test oracle, and responding with the answer from the underlying test oracle. Say X_0, X_1 are the inputs to the test oracle, and $\hat{c} = <\hat{c}_1, \hat{c}_2, \hat{c}_3, \hat{c}_4, \hat{c}_5>$ is the response.

A^* responds to queries to h_{zkp} as a normal random oracle.

Now consider a dvc $\mathsf{finish}(\eta, e, s)$ query, and let $\mathsf{start}(c)$ with $c = <c_1, c_2, c_3, c_4, c_5>$ (and $c \neq \hat{c}$) be the corresponding query initiating this instance of the protocol. A^* retrieves the ρ value generated during $\mathsf{start}(c)$ and uses it to obtain $\nu \leftarrow \eta \oplus \rho$. If $\nu^q \not\equiv_p 1$ or $e \neq h_{\mathsf{zkp}}(<\nu, (c_2)^s \nu^{-e} \bmod p, g^{s-ex_2} \bmod p>)$, then A^* simulates a dvc abort. Otherwise, A^* queries the ElGL decryption oracle with c to obtain a value z and computes the plaintext $m \leftarrow \mathsf{reveal}(z, c)$, which is then returned by A^* in response to the $\mathsf{finish}(\eta, e, s)$ query.

Finally, if A outputs a bit b for the D-ElG system, A^* outputs the same bit b for the underlying ElGL scheme.

Analysis: First we show that if $e = h_{\mathsf{zkp}}(<\nu, (c_2)^s \nu^{-e} \bmod p, g^{s-ex_2} \bmod p>)$ then with overwhelming probability $\nu \equiv_p (c_2)^{x_2}$. By the forking lemma [38], with non-negligible probability we can obtain another triple $<\nu, e', s'>$ satisfying $e' = h_{\mathsf{zkp}}(<\nu, (c_2)^{s'} \nu^{-e'} \bmod p, g^{s'-e'x_2} \bmod p>)$ where $e' \not\equiv_q e$ and

$$\begin{aligned} (c_2)^s \nu^{-e} &\equiv_p (c_2)^{s'} \nu^{-e'} \\ g^{s-ex_2} &\equiv_p g^{s'-e'x_2} \end{aligned}$$

If $\nu \equiv_p (c_2)^{x_2'}$ for some x_2', then these imply

$$\begin{aligned} s - x_2' e &\equiv_q s' - x_2' e' \\ s - x_2 e &\equiv_q s' - x_2 e' \end{aligned}$$

and thus that $(x_2' - x_2)e \equiv_q (x_2' - x_2)e'$. Since $e \not\equiv_q e'$, we must have $x_2' \equiv_q x_2$. It follows that $\nu \equiv_p (c_2)^{x_2}$ with overwhelming probability. In this case the response by the simulation for the device $\mathsf{finish}(\eta, e, s)$ is indistinguishable from the response in the real protocol, since both would return the correct plaintext (assuming $\mathsf{valid}(c) \neq 0$).

Now let D-ElG$'$ be the D-ElG protocol with f_v replaced with a perfectly random function, and let ϵ'' be the advantage of A when run against D-ElG$'$. By the pseudorandomness of f, $\epsilon'' \approx \epsilon$. Now let ϵ' be the advantage of A in the simulation, and hence the advantage of A^* in the underlying ElGL scheme. One can see that the simulation above is statistically indistinguishable from D-ElG$'$ to A, and so $\epsilon' \approx \epsilon'' \approx \epsilon$. □

Theorem 8.2 *Let h and f be random oracles. If a type* Adv($\{\mathsf{dvc}, \mathsf{svr}\}$) *attacker $(\overline{q}, \epsilon)$-breaks the* D-ElG$[\mathcal{E}, \mathcal{D}]$ *system, then there exists an attacker A^* that $(q_{\mathsf{dvc}}, \epsilon')$-breaks the underlying* ElGL *scheme with $\epsilon' \approx \epsilon - \frac{q_h + q_f}{|\mathcal{D}|}$.*

Proof: Given $A \in$ Adv($\{\mathsf{dvc}, \mathsf{svr}\}$) that $(\overline{q}, \epsilon)$-breaks the D-ElG$[\mathcal{E}, \mathcal{D}]$ system, we construct an attacker A^* for the underlying ElGL scheme. A^* is given public key $<g, p, q, y>$ for the

ElGL scheme and simulates the D-ElG system for A, such that if A succeeds in attacking D-ElG without guessing the password (as described below), A^* will succeed in attacking the underlying ElGL scheme.

Simulation: A^* gives $<g, p, q, y>$ to A as the device's public encryption key. Then A^* generates the server's key pair $(pk_{\mathsf{svr}}, sk_{\mathsf{svr}})$, and gives that to A. Next A^* generates the user password $\pi_0 \leftarrow_R \mathcal{D}$. Finally A^* generates the data $<a, b, u, g, p, q, x_2>$ for the ticket τ in the normal way, using random $t, v \in \{0,1\}^\kappa$, except that x_2 is drawn randomly from $\mathbb{Z}_q$. A^* gives a, v and $\tau = E_{pk_{\mathsf{svr}}}(<a, b, u, g, p, q, x_2>)$ to A.

A^* responds to a $h(\pi)$ or $f(v', \pi')$ query as a normal random oracle would, except that it aborts if $\pi = \pi_0$ (for an h query) or if $v' = v$ and $\pi' = \pi_0$ (for an f query). A^* responds to all other oracle queries as in the proof of Theorem 8.1. Finally, if A outputs a bit b for the D-ElG system, A^* outputs the same bit b for the underlying ElGL scheme.

Analysis: Unless A makes a query $h(\pi_0)$ or $f(v, \pi_0)$, which occurs with probability at most $\frac{q_h + q_f}{|\mathcal{D}|}$, the simulation is indistinguishable from the real protocol to A, so if A $(|\overline{q}|, \epsilon)$-breaks D-ElG, then A^* $(q_{\mathsf{dvc}}, \epsilon')$-breaks the underlying ElGL scheme, where $\epsilon' \approx \epsilon - \frac{q_h + q_f}{|\mathcal{D}|}$. □

Theorem 8.3 *Suppose h has a negligible probability of collision over $\mathcal{D}$. If a type* Adv($\{\mathsf{dvc}\}$) *attacker $(\overline{q}, \epsilon)$-breaks the* D-ElG$[\mathcal{E}, \mathcal{D}]$ *system where $\epsilon \approx \frac{q_{\mathsf{svr}}}{|\mathcal{D}|} + \psi$, then either there exists an attacker A^* that $(2q_{\mathsf{svr}}, \epsilon')$-breaks $\mathcal{E}$ with $\epsilon' \approx \frac{\psi}{2(1+q_{\mathsf{dvc}})}$ or there exists an attacker A^{**} that $(q_{\mathsf{dvc}}, \epsilon'')$-breaks the underlying* ElGL *scheme with $\epsilon'' \approx \frac{\psi}{2}$.*

Proof: Given $A \in$ Adv($\{\mathsf{dvc}\}$) that $(\overline{q}, \epsilon)$-breaks the D-ElG$[\mathcal{E}, \mathcal{D}]$ system, we show that either we can construct an attacker A^{**} for the underlying ElGL scheme, or an attacker A^* against $\mathcal{E}$. We first show that if A *wins* (as defined below) against a certain simulation with probability greater than $\frac{1}{2} + \frac{q_{\mathsf{svr}}}{|\mathcal{D}|} + \frac{\psi}{2}$ we can construct an attacker A^{**} that can $(q_{\mathsf{dvc}}, \epsilon'')$-break the underlying ElGL scheme with $\epsilon'' \approx \frac{\psi}{2}$. Assuming A does not win against that simulation with the probability stated above, then we show that we can construct an attacker A^* that $(2q_{\mathsf{svr}}, \epsilon')$-breaks $\mathcal{E}$ with $\epsilon' \approx \frac{\psi}{2(1+q_{\mathsf{dvc}})}$.

Part 1 A^{**} is given public key $<g, p, q, y>$ for the ElGL scheme and simulates the D-ElG system for A. We say A *wins* if A succeeds in guessing which of the two challenge ciphertexts X_0, X_1 was encrypted by the test oracle, or if A *makes a successful online password guess.* This is defined as A making a $\mathsf{serve}(\gamma, \delta, \tau')$ query where $\delta = \mathsf{mac}_a(<\gamma, \tau'>)$ for the mac key a stored on the device, and either (1) τ' is the ticket stored on the device and γ is a ciphertext not generated by a device start query, and where $<m, \beta, \rho> \leftarrow D_{sk_{\mathsf{svr}}}(\gamma)$, and $\beta = h(\pi_0)$; or (2) τ' is not the ticket stored on the device but γ was generated by a device start query, and where $<a', b', u', g', p', q', x_2'> \leftarrow D_{sk_{\mathsf{svr}}}(\tau')$ and $b' = h(\pi_0)$.

Part 1 Simulation: A^{**} gives $<g, p, q, y>$ to A as the device's public encryption key. Then A^{**} generates the server's key pair $(pk_{\mathsf{svr}}, sk_{\mathsf{svr}})$, and gives pk_{svr} to A. Next, A^{**} generates the user password $\pi_0 \leftarrow_R \mathcal{D}$. Finally, A^{**} generates $<a, b, u, g, p, q, x_2>$ for the ticket τ in the normal way, using $t, v \in \{0,1\}^\kappa$, except that x_2 is drawn randomly from $\mathbb{Z}_q$. A^{**} gives a, v, and $\tau \leftarrow E_{pk_{\mathsf{svr}}}(0^{5\kappa+2\lambda})$ to A.

A^{**} responds to a test oracle query, a disable oracle query, and any h_{zkp} query by A as in the proof of Theorem 8.1.

A^{**} responds to a svr disable(t', τ') query as a normal server would, but using the u value generated in the initialization if $\tau' = \tau$. A^{**} responds to serve(γ, δ, τ') queries for a τ' that has not been disabled as follows:
Case 1: (γ, δ, τ') is from a dvc start(c) query: Compute $<\nu, e, s>$ as a normal server, using x_2 generated in initialization, and c and ρ from the start(c) query.
Case 2: γ and τ' are from a dvc start query, but not δ: Have svr abort.
Case 3: $\tau' = \tau$, but γ is not from a dvc start query: Verify δ like a normal server, but using the a value from initialization as the mac key. Then compute $<c, \beta, \rho> \leftarrow D_{sk_{\mathsf{svr}}}(\gamma)$. Abort the simulation if $\beta = b$ (this is a successful online password guess), and have svr abort if $\beta \neq b$.
Case 4: $\tau' \neq \tau$, but γ is from a dvc start query: Compute $<a', b', u', g', p', q', x_2'> \leftarrow D_{sk_{\mathsf{svr}}}(\tau')$. Verify δ like a normal server, using mac key a'. Abort the simulation if $b' = b$ (this is a successful online password guess), and have svr abort if $b' \neq b$.
Case 5: $\tau' \neq \tau$ and γ is not from a dvc start query: Behave like a normal server.

A^{**} responds to a dvc start(c) query as a normal dvc would, except setting $\gamma \leftarrow E_{pk_{\mathsf{svr}}}(0^{|c|+2\kappa+2\lambda})$. A^{**} responds to a dvc finish(η, e, s) query as in the proof of Theorem 8.1.

Finally, if A outputs a bit b for the D-ElG system, A^{**} outputs the same bit b for the underlying ElGL scheme.

Part 1 Analysis: The probability that A makes a successful online password guess is at most $\frac{q_{\mathsf{svr}}}{|\mathcal{D}|}$, disregarding negligible probabilities (since π_0 was chosen randomly), so if A wins with probability at least $\frac{1}{2} + \frac{q_{\mathsf{svr}}}{|\mathcal{D}|} + \frac{\psi}{2}$, then A^{**} succeeds in distinguishing an encryption of X_0 from X_1 with probability at least $\frac{1}{2} + \frac{\psi}{2}$.

Part 2 For the second part of the proof, we assume that the probability of A winning in Part 1 is at most $\frac{1}{2} + \frac{q_{\mathsf{svr}}}{|\mathcal{D}|} + \frac{\psi}{2}$. Then we construct an attacker A^* that breaks $\mathcal{E}$ with probability $\frac{\psi}{2(1+q_{\mathsf{dvc}})}$. Our attacker A^* is given the public key pk' for $\mathcal{E}$, and runs a simulation of the D-ElG system for A.

First consider a simulator that gives pk' to A as the server's public encryption key, and then simulates D-ElG exactly, but using a decryption oracle to decrypt messages encrypted under key pk' by the adversary. There will be at most $2q_{\mathsf{svr}}$ of these. (Note that the decryptions of τ and any γ generated by the dvc would already be known to the simulator.) This simulation would be perfectly indistinguishable from the real protocol to A. Now consider the same simulation, but with the ticket and all γ values generated by the device changed to encryptions of strings of zeros. (Naturally, the server pretends the encryptions are of the normal messages, not strings of zeros.) The latter simulation is equivalent to the Part 1 simulation, except that the simulation does not abort on a successful online password guess. Still, the probability of A winning in the latter simulation is at most $\frac{1}{2} + \frac{q_{\mathsf{svr}}}{|\mathcal{D}|} + \frac{\psi}{2}$, while the probability of A winning in the former simulation is at least $\frac{1}{2} + \frac{q_{\mathsf{svr}}}{|\mathcal{D}|} + \psi$.

Now we use a standard hybrid argument to construct A^*. Let experiment $j \in \{0, \ldots, q_{\mathsf{dvc}}+1\}$ correspond to the first j ciphertexts (generated by A^*) being encryptions of the normal

messages, and the remainder being encryptions of strings of 0's, and let p_j be the probability of A forging in experiment j. Then the average value for $i \in \{0, \ldots, q_{\mathsf{dvc}}\}$ of $p_{i+1} - p_i$ is at least $\frac{\psi}{2(1+q_{\mathsf{dvc}})}$.

Therefore, to construct A^*, we simply have A^* choose a random value $i \in \{0, \ldots, q_{\mathsf{dvc}}\}$, and run experiment i as above, but calling the test oracle for $\mathcal{E}$ for the $(i+1)^{\text{st}}$ encryption it generates, 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 A^* outputs 0 if A wins (meaning A^* believes X_0 was encrypted by the test oracle), and 1 otherwise. By the analysis above, A^* breaks $\mathcal{E}$ with probability $\frac{\psi}{2(1+q_{\mathsf{dvc}})}$. □

Theorem 8.4 *If a type* $\textsc{Adv}(\{\mathsf{dvc}, \pi_0\})$ *attacker* $(\overline{q}, \epsilon)$*-breaks the* D-ElG$[\mathcal{E}, \mathcal{D}]$ *system, then either there exists an attacker* A^* *that* $(2q_{\mathsf{svr}}, \epsilon')$*-breaks* $\mathcal{E}$ *with* $\epsilon' \approx \frac{\epsilon}{2(1+q_{\mathsf{dvc}})}$ *or there exists an attacker* A^{**} *that* $(q_{\mathsf{svr}}, \epsilon'')$*-breaks the underlying* ElGL *scheme with* $\epsilon'' \approx \frac{\epsilon}{2}$.

Proof: Given $A \in \textsc{Adv}(\{\mathsf{dvc}, \pi_0\})$ that $(\overline{q}, \epsilon)$-breaks the D-ElG$[\mathcal{E}, \mathcal{D}]$ system, we show that either we can construct an attacker A^{**} for the underlying ElGL scheme, or an attacker A^* against $\mathcal{E}$. We first show that if A succeeds against a certain simulation with probability greater than $\frac{1}{2} + \frac{\epsilon}{2}$, we can construct an attacker A^{**} that can $(q_{\mathsf{svr}}, \epsilon')$-break the underlying ElGL scheme with $\epsilon' \approx \frac{\epsilon}{2}$. Assuming A does not succeed against that simulation with the probability stated above, then we show that we can construct an attacker A^* that breaks $\mathcal{E}$ with probability at least $\frac{\epsilon}{2(1+q_{\mathsf{dvc}})}$.

Part 1 A^{**} is given $<g, p, q, y>$ for the ElGL encryption scheme and simulates the D-ElG system for A.

Part 1 Simulation: A^{**} gives $<g, p, q, y>$ to A as the device's public signature key. Then A^{**} generates the server's key pair $(pk_{\mathsf{svr}}, sk_{\mathsf{svr}})$, and gives pk_{svr} to A. Next A^{**} generates $\pi_0 \leftarrow_R \mathcal{D}$ and gives π_0 to A. Finally, A^{**} generates a, b, u, and x_1 in the normal way, using random $t, v \in \{0,1\}^\kappa$, but computes $\tau \leftarrow E_{pk_{\mathsf{svr}}}(0^{5\kappa+2\lambda})$. A^{**} gives a, v, and τ to A.

A^{**} responds to a test oracle query, a disable oracle query, an h_{zkp} query, and a $\mathsf{svr}\ \mathsf{disable}(t', \tau')$ query as as in the proof of Theorem 8.3 (Part 1 Simulation).

A^{**} responds to $\mathsf{svr}\ \mathsf{serve}(\gamma, \delta, \tau')$ queries for a τ' that has not been disabled as follows:
Case 1: (γ, δ, τ') is from a $\mathsf{dvc}\ \mathsf{start}(c)$ query: Retrieve c and ρ from the $\mathsf{start}(c)$ query. Query the decryption oracle with c to get z, compute $w \leftarrow \mathsf{select}(c)$, $\nu \leftarrow zw^{-x_1} \bmod p$, $e \leftarrow_R \mathbb{Z}_q$, and $s \leftarrow_R \mathbb{Z}_q$, and "backpatch" h_{zkp}:

$$h_{\mathsf{zkp}}(<\nu, (c_2)^s \nu^{-e} \bmod p, g^s (y g^{-x_1})^{-e} \bmod p>) \quad \leftarrow \quad e$$

Return $<\rho \oplus \nu, e, s>$.
Case 2: γ and τ' are from a $\mathsf{dvc}\ \mathsf{start}$ query, but not δ: Have svr abort.
Case 3: $\tau' = \tau$, but γ is not from a $\mathsf{dvc}\ \mathsf{start}$ query: Verify δ like a normal server, but using the a value from initialization as the mac key. Then compute $<c, \beta, \rho> \leftarrow D_{sk_{\mathsf{svr}}}(\gamma)$. Have svr abort if $\beta \neq b$ or $\mathsf{valid}(c) = 0$. Otherwise, query the decryption oracle with c to get z, compute $w \leftarrow \mathsf{select}(c)$, $\nu \leftarrow zw^{-x_1} \bmod p$, $e \leftarrow_R \mathbb{Z}_q$, and $s \leftarrow_R \mathbb{Z}_q$, and "backpatch" h_{zkp}:

$$h_{\mathsf{zkp}}(<\nu, (c_2)^s \nu^{-e} \bmod p, g^s (y g^{-x_1})^{-e} \bmod p>) \quad \leftarrow \quad e$$

Return $<\rho \oplus \nu, e, s>$.
Case 4: $\tau' \neq \tau$, but γ is from a dvc start query: Compute $<a', b', u', g', p', q', x'_2> \leftarrow D_{sk_{\mathsf{svr}}}(\tau')$. Behave like a normal server, but using c, β, and ρ from the dvc start(c) query.
Case 5: $\tau' \neq \tau$ and γ is not from a dvc start query: Behave like a normal server.

A^{**} responds to a start(c) query as a normal dvc would, except setting $\gamma \leftarrow E_{pk_{\mathsf{svr}}}(0^{|c|+2\kappa+2\lambda})$. For a finish$(\eta, e, s)$ query corresponding to a start(c) query that returned (γ, δ, τ), if $<\eta, e, s>$ was not returned from a svr serve(γ, δ, τ) query, have dvc abort. Otherwise for the z value found in that serve(γ, δ, τ) query, A^{**} returns the plaintext $m \leftarrow$ reveal(z, c).

Part 1 Analysis: If A distinguishes an encryption of X_0 from X_1 with probability at least $\frac{1}{2} + \frac{\epsilon}{2}$ in the simulation, A^{**} distinguishes an encryption of X_0 from X_1 with probability at least $\frac{1}{2} + \frac{\epsilon}{2}$ in the underlying ElGL encryption scheme.

Part 2 The second part of the proof is similar to Part 2 of the proof of Theorem 8.3, except with ψ replaced by ϵ, and no $\frac{q_{\mathsf{svr}}}{|\mathcal{D}|}$ term. □

9 Conclusion

Dictionary attacks against password-protected private keys are a significant threat if the device holding those keys may be captured. In this paper we have presented an approach to render devices invulnerable to such attacks. Our approach requires the device to interact with a remote server to perform its private key operations. Therefore, it is primarily suited to a device that uses its private key in interactive cryptographic protocols (and so necessarily has network connectivity to reach the server when use of its private key is required). A prime example is a device that plays the role of a client in the TLS protocol with client authentication. Though our protocol requires the device to interact with a remote server, we prove that this server poses no threat to the device. Specifically, it gains no significant advantage in forging signatures that can be verified with the device's public key or decrypting messages encrypted under the device's public key. In particular, it cannot mount a dictionary attack to expose the device's private key. Even if *both* the device and server are compromised, the attacker must still succeed in an offline dictionary attack before signing on behalf of the device.

In addition to the above properties, we presented protocols that further provide the feature of *key disabling*. This enables the user to disable the device's private key immediately, even after the device has been captured and even if the attacker has guessed the user's password. Once disabled, the device's key is provably useless to the attacker (provided that the attacker cannot also compromise the server). Key disabling is thus an effective complement to any public key revocation mechanism that might exist, particularly if there is a delay for revoking public keys.

References

[1] N. Asokan, G. Tsudik, and M. Waidner. Server-supported signatures. *Journal of Computer Security* 5(1), 1997.

[2] P. Béguin and J. J. Quisquater. Fast server-aided RSA signatures secure against active attacks. In *Advances in Cryptology—CRYPTO '95* (Lecture Notes in Computer Science 963), pp. 57–69, 1995.

[3] M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations among notions of security for public-key encryption schemes. In *Advances in Cryptology—CRYPTO '98* (Lecture Notes in Computer Science 1462), pp. 26–45, 1998.

[4] M. Bellare and S. Miner. A forward-secure digital signature scheme. In *Advances in Cryptology—CRYPTO '99* (Lecture Notes in Computer Science 1666), pp. 431–438, 1999.

[5] M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In *Advances in Cryptology—EUROCRYPT 2000* (Lecture Notes in Computer Science 1807), pp. 139–155, 2000.

[6] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In 1st *ACM Conference on Computer and Communications Security*, pp. 62–73, Nov. 1993.

[7] M. Bellare and P. Rogaway. Optimal asymmetric encryption. In *Advances in Cryptology—EUROCRYPT '94* (Lecture Notes in Computer Science 950), pp. 92–111, 1995.

[8] M. Bellare and P. Rogaway. The exact security of digital signatures—How to sign with RSA and Rabin. In *Advances in Cryptology—EUROCRYPT '96* (Lecture Notes in Computer Science 1070), pp. 399–416, 1996.

[9] S. M. Bellovin and M. Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *1992 IEEE Symposium on Security and Privacy*, pp. 72–84, 1992.

[10] C. Boyd. Digital multisignatures. In H. J. Beker and F. C. Piper, editors, *Cryptography and Coding*, pp. 241–246. Clarendon Press, 1989.

[11] V. Boyko, P. MacKenzie, and S. Patel. Provably secure password authentication and key exchange using Diffie-Hellman. In *Advances in Cryptology—EUROCRYPT 2000* (Lecture Notes in Computer Science 1807), pp. 156–171, 2000.

[12] R. Canetti, O. Goldreich and S. Halevi. The random oracle methodology, revisited. In 30th *ACM Symposium on Theory of Computing*, pp. 209–218, 1998.

[13] R. Cramer and V. Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *Advances in Cryptology—CRYPTO '98* (Lecture Notes in Computer Science 1462), pp. 13–25, 1998.

[14] D. E. Denning. Digital signatures with RSA and other public-key cryptosystems. *Communications of the ACM* 27(4):388–392, Apr. 1984.

[15] T. Dierks and C. Allen. The TLS protocol version 1.0. IETF Request for Comments 2246, Jan. 1999.

[16] D. Dean, T. Berson, M. Franklin, D. Smetters and M. Spreitzer. Cryptography as a network service. In *2001 ISOC Symposium on Network and Distributed System Security*, Feb. 2001.

[17] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory* 31:469–472, 1985.

[18] D. Feldmeier and P. Karn. UNIX password security—Ten years later. In *Advances in Cryptology—CRYPTO '89* (Lecture Notes in Computer Science 435), 1990.

[19] W. Ford and B. S. Kaliski, Jr. Server-assisted generation of a strong secret from a password. In 5th *IEEE International Workshop on Enterprise Security*, 2000.

[20] R. Ganesan. Yaksha: Augmenting Kerberos with public key cryptography. In *Proceedings of the 1995 ISOC Network and Distributed System Security Symposium*, Febrary 1995.

[21] O. Goldreich, S. Goldwasser and S. Micali. How to construct random functions. *Journal of the ACM* 33(4):792–807, Oct. 1984.

[22] S. Goldwasser, S. Micali, and R. L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal of Computing* 17(2):281–308, Apr. 1988.

[23] J. Håstad, J. Honsson, A. Juels, and M. Yung. Funkspiel schemes: An alternative to conventional tamper resistance. In 7th *ACM Conference on Computer and Communications Security*, pp. 125–133, Nov. 2000.

[24] S. Halevi and H. Krawczyk. Public-key cryptography and password protocols. In 5th *ACM Conference on Computer and Communications Security*, pp. 122–131, 1998.

[25] D. N. Hoover and B. N. Kausik. Software smart cards via cryptographic camouflage. In *1999 IEEE Symposium on Security and Privacy*, pp. 208–215, May 1999.

[26] S. Hong, J. Shin, H. Lee-Kwang, and H. Yoon. A new approach to server-aided secret computation. In 1st *International Conference on Information Security and Cryptology*, pp. 33–45, 1998.

[27] D. Jablon. Strong password-only authenticated key exchange. *ACM Computer Communication Review* 26(5):5–20, 1996.

[28] D. Klein. Foiling the cracker: A survey of, and improvements to, password security. In 2nd *USENIX Security Workshop*, Aug. 1990.

[29] D. W. Kravitz. Digital signature algorithm. U.S. Patent 5,231,668, 27 July 1993.

[30] H. Krawczyk. Simple forward-secure signatures from any signature scheme. In 7th *ACM Conference on Computer and Communication Security*, pp. 108–115, Nov. 2000.

[31] T. M. A. Lomas, L. Gong, J. H. Saltzer, and R. M. Needham. Reducing risks from poorly chosen keys. *ACM Operating Systems Review* 23(5):14–18, Dec. 1989.

[32] P. MacKenzie, S. Patel, and R. Swaminathan. Password authenticated key exchange based on RSA. In *Advances in Cryptology—ASIACRYPT 2000*, pp. 599–613, 2000.

[33] P. MacKenzie and M. K. Reiter. Two-party generation of DSA signatures. In *Advances in Cryptology—CRYPTO 2001*, 2001. To appear.

[34] T. Matsumoto, K. Kato, and H. Imai. Speeding up computation with insecure auxiliary devices. In *Advances in Cryptology—CRYPTO '88* (Lecture Notes in Computer Science 403), pp. 497–506, 1989.

[35] U. Maurer and S. Wolf. The Diffie-Hellman protocol. *Designs, Codes, and Cryptography* 19:147–171, Kluwer Academic Publishers, 2000.

[36] R. Morris and K. Thompson. Password security: A case history. *Communications of the ACM*, 22(11):594–597, Nov. 1979.

[37] R. Perlman and C. Kaufman. Secure password-based protocol for downloading a private key. In *1999 Network and Distributed System Security Symposium*, Feb. 1999.

[38] D. Pointcheval and J. Stern. Security proofs for signature schemes. In *Advances in Cryptology—EUROCRYPT '96* (Lecture Notes in Computer Science 1070), pages 387–398, 1996.

[39] C. Rackoff and D. Simon. Noninteractive zero-knowledge proof of knowledge and chosen ciphertext attack. In *Advances in Cryptology—CRYPTO '91*, pp. 433–444, 1991.

[40] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM* 21(2):120–126, Feb. 1978.

[41] V. Shoup and R. Gennaro. Securing threshold cryptosystems against chosen ciphertext attack. In *Advances in Cryptology—EUROCRYPT '98*, pp. 1–16, 1998.

[42] T. Wu. The secure remote password protocol. In *1998 Network and Distributed System Security Symposium*, Feb. 1999.

[43] T. Wu. A real-world analysis of Kerberos password security. In *1999 Network and Distributed System Security Symposium*, Feb. 1999.