DIMACS Technical Report 2001-37
November 2001

Delegation of Cryptographic Servers for Capture-Resilient Devices

by

Philip MacKenzie[1]
Bell Labs, Lucent Technologies
Murray Hill, NJ, USA
philmac@research.bell-labs.com

Michael K. Reiter[2]
Carnegie Mellon University
Pittsburgh, PA, USA
reiter@cmu.edu

[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

A device that performs private key operations (signatures or decryptions), and whose private key operations are protected by a password, can be immunized against offline dictionary attacks in case of capture by forcing the device to confirm a password guess with a designated remote server in order to perform a private key operation. Recent proposals for achieving this allow untrusted servers and require no server initialization per device. In this paper we extend these proposals to enable dynamic delegation from one server to another; i.e., the device can subsequently use the second server to secure its private key operations. One application is to allow a user who is traveling to a foreign country to temporarily delegate to a server local to that country the ability to confirm password guesses and aid the user's device in performing private key operations, or in the limit, to temporarily delegate this ability to a token in the user's possession. Another application is proactive security for the device's private key, i.e., proactive updates to the device and servers to eliminate any threat of offline password guessing attacks due to previously compromised servers.

1 Introduction

A device that performs private key operations (signatures or decryptions) risks exposure of its private key if captured. While encrypting the private key with a password is common, this provides only marginal protection, since passwords are well-known to be susceptible to offline dictionary attacks (e.g., [16, 13]). Much recent research has explored better password protections for the private keys on a device that may be captured. These include techniques to encrypt the private key under a password in a way that prevents the attacker from verifying a successful password guess (*cryptographic camouflage*) [12]; or to force the attacker to verify his password guesses at an online server, thereby turning on offline attack into an online one that can be detected and stopped (e.g., [9, 14]).

We take as our starting point the latter approach, in which an attacker who captures that device must validate its password guesses at a remote server before the use of the private key is enabled. In particular, we focus on the proposals of [14], in which this server 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. This approach offers certain advantages: e.g., it is compatible with existing infrastructure, whereas cryptographic camouflage requires that "public" keys be hidden from potential attackers. However, it also comes with the disadvantage that the device must interact with a designated remote server in order to perform a (and typically each) private key operation. This interaction may become a bottleneck if the designated remote server is geographically distant and the rate of private key operations is significant.

In this paper, we investigate a technique to alleviate this limitation, with which a device may temporarily delegate the password-checking function from its originally designated server to another server that is closer to it. For example, a traveler in a foreign country may temporarily delegate the password-checking function for her laptop computer to a server in the country she is visiting. By doing so, her device's subsequent private key operations will require interaction only with this local server, presumably incurring far less latency than if the device were interacting with the original server. In the limit, the user could temporarily delegate to a hardware token in her possession, so that the device could produce signatures or decryptions in offline mode without network access at all.

Of course, delegating the password-checking function from one server to another has security implications. As originally developed, the techniques that serve as our starting point [14] have the useful property that the designated server, in isolation, gains no information that would enable it to forge signatures or decrypt ciphertexts on the device's behalf. However, if both it and the device were captured, then the attacker could mount an offline dictionary attack against the password, and then forge signatures or decrypt ciphertexts for the device if he succeeds. Naturally, in the case of delegation, this vulnerability should not extend to any server *ever* delegated by the device. Rather, our high-level security goal is to ensure that an individual server authorized for password-checking by delegation, and whose authority is then revoked, poses the same security threat as a server to which delegation never occurred in the first place. Specifically, an attacker that captures the device after the device has revoked the authorization of a server (even if the server was previously compromised) must still conduct an *online* dictionary attack at an authorized server in order to attack the password.

Even with this goal achieved, delegation does impinge on security in at least two ways, however. First, if the attacker captures the device, then it can mount an online dictionary attack against *each* currently authorized server, thereby gaining more password guesses than any one server allows. Second, a feature of the original protocols is that the password-checking server could be permanently *disabled* for the device even after the device *and* password were compromised; by doing so, the device can never sign or decrypt again. In a system supporting delegation, however, if the device and password are compromised, and if there is some authorized server when this happens, then the attacker can delegate from this authorized server to *any* server permitted by the policy set forth when the device was initialized. Thus, to be sure that the device will never sign or decrypt again, every server in this permissible set must be disabled for the device.

As a side effect of achieving our security goals, our techniques offer a means for realizing *proactive security* (e.g., [11]) in the context of [14]. Intuitively, proactive security encompasses techniques for periodically refreshing the cryptographic secrets held by various components of a system, thereby rendering any cryptographic secrets captured before the refresh useless to the attacker. Our delegation protocol can be used as a subroutine for proactively refreshing a password-checking server, so that if the server's secrets had been exposed, they are useless to the attacker after the refresh. In particular, if the attacker subsequently captured the device, any dictionary attack that the attacker could mount would be *online*, as opposed to offline.

In this paper we specify security requirements for delegation in this context and then describe delegation systems for RSA signing [20] and ElGamal decryption [8]. Supporting delegation for these not only requires devising custom delegation protocols for RSA and ElGamal keys, but also modifying the original signing and decryption protocols of [14] to accommodate delegation. For example, our revised RSA system utilizes three-way function sharing, versus the two-way function sharing used in the original system; this seems to be required to accomplish our objectives. And, whereas the original systems of [14] permitted the server to conduct an offline dictionary attack against the user's password (without placing the device's signing key at risk), here we must prevent a server from conducting such an attack. Our delegation protocols themselves also contribute points of technical interest, as we will discuss later.

An extended abstract of this paper [15] presented an RSA signing system supporting delegation with weaker security properties than those described here. That version also did not describe an ElGamal decryption system supporting delegation, as we do here.

2 Preliminaries

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

2.1 System model

Our system consists of a device dvc and an arbitrary, possibly unknown, number of servers. A server will be denoted by svr, possibly with subscripts or other annotations when useful. The device communicates to a server over a public network. In our system, the device is used either for generating signatures or decrypting messages, and does so by interacting with one of the servers. 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 each of the servers. The public and secret data associated with a server should simply be a certified public key and associated private key for the server, 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 one of the servers. This server must be *authorized* to execute this protocol. (We define *authorized* precisely below.) The system is initialized with exactly one server authorized, denoted svr_0. Further servers may be authorized, but this authorization cannot be performed by dvc alone. Rather, for dvc to authorize svr, another already-authorized server svr' must also consent to the authorization of svr after verifying that the authorization of svr is consistent with policy previously set forth by dvc and is being performed by dvc with the user's password. In this way, authorization is a protected operation just as signing and decryption are. The device can unilaterally *revoke* the authorization of a server when it no longer intends to use that server. A server can be *disabled* (for a device) by being instructed to no longer respond to that device or, more precisely, to requests involving its key.

For the purposes of this paper, the aforementioned policy dictating which servers can be authorized is expressed as a set U of servers with well-known public keys. That is, an authorized server svr will consent to authorize another server svr' only if $\mathsf{svr}' \in U$. Moreover, we assume that svr can reliably determine the unique public key $pk_{\mathsf{svr}'}$ of any $\mathsf{svr}' \in U$. In practice, this policy would generally need to be expressed more flexibly; for example, a practical policy might allow any server with a public key certified by a given certification authority to be authorized. For such a policy, our delegation protocols would then need to be augmented with the appropriate certificates and certificate checks; for simplicity, we omit such details here.

To specify security for our system, we must consider the possible attackers that attack the system. Each attacker we consider in this paper is presumed to control the network; i.e., the attacker controls the inputs to the device and every server, and observes the outputs. Moreover, an attacker can permanently *compromise* certain resources. The possible resources that may be compromised by the attacker are any of the servers, dvc, and π_0. Compromising reveals the entire contents of the resource to the attacker. The one restriction on the attacker is that if he compromises dvc, then he does so after dvc initialization and while dvc is in an inactive state—i.e., dvc is not presently executing a 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 formalize the aspects of the system described thus far as a collection of *events*.

1. $\mathsf{dvc.startDel}(\mathsf{svr}, \mathsf{svr}')$: dvc begins a delegation protocol with server svr to authorize svr'.

2. $\mathsf{dvc.finishDel}(\mathsf{svr}, \mathsf{svr}')$: dvc finishes a delegation protocol with server svr to authorize svr'. This can occur only after a $\mathsf{dvc.startDel}(\mathsf{svr}, \mathsf{svr}')$ with no intervening $\mathsf{dvc.finishDel}(\mathsf{svr}, \mathsf{svr}')$, $\mathsf{dvc.revoke}(\mathsf{svr})$ or $\mathsf{dvc.revoke}(\mathsf{svr}')$.

3. $\mathsf{dvc.revoke}(\mathsf{svr})$: dvc revokes the authorization of svr.

4. $\mathsf{svr.disable}$: svr stops responding to any requests of the device (signing, decryption, or delegation).

5. $\mathsf{dvc.comp}$: dvc is compromised (and captured).

6. $\mathsf{svr.comp}$: svr is compromised.

7. $\pi_0.\mathsf{comp}$: the password π_0 is compromised.

The time of any event x is given by $T(x)$. Now we define the following predicates for any time t:

- $\mathsf{authorized}_t(\mathsf{svr})$ is true iff either (i) $\mathsf{svr} = \mathsf{svr}_0$ and there is no $\mathsf{dvc.revoke}(\mathsf{svr}_0)$ prior to time t, or (ii) there exist a svr' and event $x = \mathsf{dvc.finishDel}(\mathsf{svr}', \mathsf{svr})$ where $\mathsf{authorized}_{T(x)}(\mathsf{svr}')$ is true, $T(x) < t$, and no $\mathsf{dvc.revoke}(\mathsf{svr})$ occurs between $T(x)$ and t. In case (ii), we call svr' the *consenting server*.

- $\mathsf{nominated}_t(\mathsf{svr})$ is true iff there exist a svr' and event $x = \mathsf{dvc.startDel}(\mathsf{svr}', \mathsf{svr})$ where $\mathsf{authorized}_{T(x)}(\mathsf{svr}')$ is true, $T(x) < t$, and none of $\mathsf{dvc.finishDel}(\mathsf{svr}', \mathsf{svr})$, $\mathsf{dvc.revoke}(\mathsf{svr})$, or $\mathsf{dvc.revoke}(\mathsf{svr}')$ occur between $T(x)$ and t.

For any event x, let

$$\mathsf{Active}(x) \;=\; \{\ \mathsf{svr} : \mathsf{nominated}_{T(x)}(\mathsf{svr}) \vee \mathsf{authorized}_{T(x)}(\mathsf{svr})\ \}$$

2.2 Goals

It is convenient in specifying our security goals to partition attackers into four classes, depending on the resources they compromise and the state of executions when these attackers compromise certain resources. An attacker is assumed to fall into one of these classes independent of the execution, i.e., it does not change its behavior relative to these classes depending on the execution of the system. In particular, the resources an attacker compromises are assumed to be independent of the execution. In this sense, we consider static attackers only (in contrast to adaptive ones).

A1. An attacker in class A1 does not compromise dvc or compromises dvc only if $\mathsf{Active}(\mathsf{dvc.comp}) = \emptyset$.

A2. An attacker in class A2 is not in class A1, does not compromise π_0, and compromises dvc only if $\mathsf{svr.comp}$ never occurs for any $\mathsf{svr} \in \mathsf{Active}(\mathsf{dvc.comp})$.

A3. An attacker in class A3 is not in class A1, does not compromise π_0, and compromises dvc only if $\mathsf{svr.comp}$ occurs for some $\mathsf{svr} \in \mathsf{Active}(\mathsf{dvc.comp})$.

A4. An attacker in class A4 is in none of classes A1, A2, or A3, and does not compromise any $\mathsf{svr} \in U$.

Now we state the security goals of our systems against these attackers as follows (disregarding events that occur with negligible probability):

G1. An A1 attacker is unable to forge signatures or decrypt messages for dvc.

G2. An A2 attacker can forge signatures or decrypt messages for the device with probability at most $\frac{q}{|\mathcal{D}|}$, where q is the total number of queries to servers in $\mathsf{Active}(\mathsf{dvc.comp})$ after $T(\mathsf{dvc.comp})$, and $\mathcal{D}$ is the dictionary from which the password is drawn (at random).

G3. An A3 attacker can forge signatures or decrypt messages for the device only if it succeeds in an offline dictionary attack on the password.

G4. An A4 attacker can forge signatures or decrypt messages only until $\max_{\mathsf{svr} \in U}\{T(\mathsf{svr.disable})\}$.

These goals can be more intuitively stated as follows. First, if an attacker does not capture dvc, or does so only when no servers are authorized for dvc (A1), then the attacker gains no ability to forge or decrypt for the device (G1). On the other extreme, if an attacker captures both dvc and π_0 (A4)—and thus is indistinguishable from the user—it can forge only until all servers are disabled (G4) or indefinitely if it also compromises a server.[1] The "middle" cases are if the attacker compromises dvc and not π_0. If it compromises dvc and no then-authorized server is ever compromised (A2), then the attacker can do no better than an online dictionary attack against π_0 (G2). If, on the other hand, when dvc is compromised some authorized server is eventually compromised (A3), then the attacker can do no better than an offline attack against the password (G3). It is not difficult to verify that these goals are a strict generalization of the goals of [14]; i.e., these goals reduce to those of [14] in the case $|U| = 1$.

These properties offer an approach to proactively update dvc to render useless to an attacker any information it gained by compromising a server. That is, suppose that each physical computer running a logical server svr periodically instantiates a new logical server svr' having a new public and private key. If dvc delegates from svr to svr', and if then dvc revokes svr, any disclosure of information from svr (e.g., the private key of svr) is then useless for the attacker in its efforts to forge or decrypt for dvc. Rather, if the attacker captures dvc, it must compromise svr' in order to conduct an offline attack against dvc.

[1] If the attacker compromises a server along with both dvc and π_0, then it can delegate from any authorized server to that compromised server to forge or decrypt indefinitely.

2.3 Tools

Our systems for meeting the goals outlined in Section 2.2 utilize a variety of cryptographic tools, which we define informally below.

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 systems meet every goal outlined above, we generally require that these hash functions behave like random oracles [2]. (For heuristics on instantiating random oracles, see [2].) 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.

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 $c \leftarrow E_{pk}(m)$, and we denote the maximum bit length of c by $\ell(\lambda)$. 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 under the corresponding public key, if such an m exists, and otherwise returns $\perp$. Our systems require an encryption scheme secure against adaptive chosen ciphertext attacks [19]. Practical examples can be found in [3, 6].

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 for the corresponding private key, and otherwise returns the bit $b = 0$. That is, $b \leftarrow V(pk, m, \sigma')$. Naturally, if $\sigma \leftarrow S(sk, m)$, then $V(pk, m, \sigma) = 1$.

3 Delegation for S-RSA

The work on which this paper is based [14] described several systems by which dvc could involve a server for performing the password-checking function and assisting in its cryptographic operations, and thereby gain immunity to offline dictionary attacks if captured. The first of these systems, denoted GENERIC, did not support the disabling property (the instantiation of G4 for a single server and no delegation), but worked for any type of public key algorithm that dvc used. As part of the signing/decryption protocol in this system, dvc recovered the private key corresponding to its public key. This, in turn, renders delegation in this system straightforward, being roughly equivalent to a re-initialization of the device using the same private key, but for a different server. The few minor technical changes needed to accommodate delegation are also reflected in the RSA system we detail here, and so we omit further discussion of GENERIC.

The system described in [14] by which dvc performs RSA signatures is called S-RSA. At a high level, S-RSA uses 2-out-of-2 function sharing to distribute the ability to generate a signature for the device's public key between the device and the server. The server, however, would cooperate with the device to sign a message only after being presented with evidence that the device was in possession of the user's correct password.

In this section we describe a new system for RSA signatures, called S-RSA-DEL, that supports delegation in addition to signatures. In order to accommodate delegation in this context, the system is changed so that the device's signature function is shared using a 3-out-of-3 function sharing, where one of the three shares is generated from the password itself. In this way, the user share (i.e., the password) may remain the same while the device share is changed for delegation purposes. Also, the server share is split into an encrypted part generated by the consenting server and an unencrypted part generated by the device, to allow the device to construct a new ticket in which neither the device nor the consenting server can determine the new server's share. Other changes are needed as well; e.g., whereas the server in the S-RSA system could mount an offline dictionary attack against the user's password (without risk to the device's signature operations), here we must prevent the server from mounting such an attack. While introducing these changes to the signing protocol, and introducing the new delegation protocol, we strive to maintain the general protocol structure of S-RSA.

3.1 Preliminaries

We suppose the device creates signatures using a standard encode-then-sign RSA signature algorithm (e.g., "hash-and-sign" [7]). The public and secret keys of the device are $pk_{\mathsf{dvc}} = {<}e, N{>}$ and $sk_{\mathsf{dvc}} = {<}d, N, \phi(N){>}$, respectively, 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):

$$\begin{aligned} S(<d, N, \phi(N)>, m){:}\ & r \leftarrow_R \{0,1\}^{\kappa_{sig}} \\ & \sigma \leftarrow (\mathsf{encode}(m, r))^d \bmod N \\ & \text{return } <\sigma, r> \end{aligned}$$

Note that it may not be necessary to return r if it can be determined from m and σ. 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 [4] 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 [2, 4]. Naturally any signature of this form can be verified by checking that $\sigma^e \equiv_N \mathsf{encode}(m, r)$. In the function sharing primitive used in our system, d is broken into shares d_0, d_1 and d_2 such that $d_0 + d_1 + d_2 \equiv_{\phi(N)} d$ [5].

3.2 Device initialization

The inputs to device initialization are the identity of svr_0 and its public encryption key pk_{svr_0}, 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:

$$\begin{aligned} & t \leftarrow_R \{0,1\}^{\kappa} \\ & u \leftarrow h_{\mathsf{dsbl}}(t) \\ & v \leftarrow_R \{0,1\}^{\kappa} \\ & a \leftarrow_R \{0,1\}^{\kappa} \\ & b \leftarrow f(v, \pi_0) \\ & d_0 \leftarrow h(\pi_0) \\ & d_1 \leftarrow_R \{0,1\}^{\lambda+\kappa} \\ & d_2 \leftarrow d - d_1 - d_0 \bmod \phi(N) \\ & \zeta \leftarrow E_{pk_{\mathsf{svr}_0}}(<u, d_2>) \\ & \tau \leftarrow E_{pk_{\mathsf{svr}_0}}(<a, b, 0, \zeta, N>) \end{aligned}$$

h is assumed to output a $(\lambda + \kappa)$-bit value. The values pk_{dvc}, u and *authorization record* $<\mathsf{svr}_0, pk_{\mathsf{svr}_0}, \tau, t, v, d_1, a>$ are saved on stable storage in the device. All other values, including d, $\phi(N)$, π_0, b, d_0, d_2, and ζ, are deleted from the device. The value t should be backed up offline for use in disabling if the need arises. The τ value is the device's "ticket" that it uses to access svr_0. The u value is the "ticket identifier".

The ticket τ will be sent to svr within the context of the S-RSA-DEL signing and delegation protocols (see Sections 3.3 and 3.4), and the server will inspect the contents of the ticket to extract its share d_2 of the device's private signing key. In anticipation of its own compromise, dvc might include a policy statement within τ to instruct svr_0 as to what it should or should not do with requests bearing this ticket. This policy could include an intended expiration time for τ, instructions to cooperate in signing messages only of a certain form, or instructions to cooperate in delegating only to certain servers. As discussed in Section 2.1, here we assume a default policy that restricts delegation to only servers in U. For simplicity, we omit this policy and its inspection from device initialization and subsequent protocols, but a practical implementation must support it.

3.3 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 the identity svr of the server to be used, such that dvc holds an authorization record $<\mathsf{svr}, pk_{\mathsf{svr}}, \tau, t, v, d_1, a>$, generated either in the initialization procedure of Section 3.2, or in the delegation protocol of Section 3.4. Recall that dvc also stores $pk_{\mathsf{dvc}} = <e, N>$. In this protocol, and all following protocols, we do not explicitly check that message parameters are of the correct form and fall within the appropriate bounds, but any implementation must do this. The protocol is described in Figure 1.

This protocol generates a signature for m by constructing $\mathsf{encode}(m, r)^{d_0+d_1+d_2} \bmod N$, where d_0 is derived from the user's password, d_1 is stored on dvc, and d_2 is stored (partially encrypted) in τ. $\nu = \mathsf{encode}(m, r)^{d_2} \bmod N$ is computed at svr after svr has confirmed that the β value contained within the ciphertext γ is valid evidence that dvc holds the user's password. The device multiples ν by $\mathsf{encode}(m, r)^{d_0+d_1} \bmod N$ to get the desired result. It is important that the device delete β, d_0 and ρ (used to encrypt ν) when the protocol completes, and that it never store them on stable storage.

δ is a message authentication code computed using a, to show the server that this request originated from the device. δ enables svr to distinguish an incorrect password guess by someone holding the device from a request created by someone not holding the device. Since svr should respond to only a limited number of the former (lest it allow an online dictionary attack to progress too far), δ is important in preventing denial-of-service attacks against the device by an attacker who has not compromised the device.

3.4 Delegation protocol

Here we present the protocol by which the device delegates the capability to help it perform cryptographic operations to a new server. The inputs provided to the device are the identity svr of the server to be used, such that dvc holds an authorization record $<\mathsf{svr}, pk_{\mathsf{svr}}, \tau, t, v, d_1, a>$, a public key $pk_{\mathsf{svr}'}$ for another server $\mathsf{svr}' \in U$, and the input password π. (As described in Section 3.2, one could also input additional policy information here.) Recall that dvc also stores $pk_{\mathsf{dvc}} = <e, N>$. The protocol is described in Figure 2. In this figure, h_{dele} is assumed to output a $(\lambda + \kappa)$-bit value.

The overall goal of the protocol in Figure 2 is to generate a new share d_2' for server svr′, and new share d_1' and new ticket τ' for the device to use with svr′. The device's new share d_1' is created as the sum of d_{11}' and d_{21}', selected randomly by dvc and svr, respectively. The

dvc		svr
$\beta \leftarrow f(v, \pi)$		
$\rho \leftarrow_R \{0,1\}^{\lambda}$		
$r \leftarrow_R \{0,1\}^{\kappa_{sig}}$		
$\gamma \leftarrow E_{pk_{\mathsf{svr}}}(<m, r, \beta, \rho>)$		
$\delta \leftarrow \mathsf{mac}_a(<\gamma, \tau>)$		
	$\xrightarrow{\gamma,\delta,\tau}$	
		$<a, b, d_{12}, \zeta, N> \leftarrow D_{sk_{\mathsf{svr}}}(\tau)$
		abort if $\mathsf{mac}_a(<\gamma, \tau>) \neq \delta$
		$<m, r, \beta, \rho> \leftarrow D_{sk_{\mathsf{svr}}}(\gamma)$
		abort if $\beta \neq b$
		$<u, d_{22}> \leftarrow D_{sk_{\mathsf{svr}}}(\zeta)$
		abort if u is disabled
		$d_2 \leftarrow d_{12} + d_{22}$
		$\nu \leftarrow (\mathsf{encode}(m, r))^{d_2} \bmod N$
		$\eta \leftarrow \rho \oplus \nu$
	$\xleftarrow{\eta}$	
$\nu \leftarrow \rho \oplus \eta$		
$d_0 \leftarrow h(\pi)$		
$\sigma \leftarrow \nu(\mathsf{encode}(m, r))^{d_0 + d_1} \bmod N$		
abort if $\sigma^e \not\equiv_N \mathsf{encode}(m, r)$		
return $<\sigma, r>$		

Figure 1: S-RSA-Del signature protocol

dvc		svr
$\alpha \leftarrow_R \{0,1\}^\kappa$ $\beta \leftarrow f(v,\pi)$ $\rho \leftarrow_R \{0,1\}^{\lambda+\kappa+\ell(\lambda)}$ $\gamma \leftarrow E_{pk_{\mathsf{svr}}}(<\beta, pk_{\mathsf{svr}'}, \rho, \alpha>)$ $\delta \leftarrow \mathsf{mac}_a(<\gamma,\tau>)$		
	$\xrightarrow{\gamma,\delta,\tau}$	
		$<a,b,d_{12},\zeta,N> \leftarrow D_{sk_{\mathsf{svr}}}(\tau)$ abort if $\mathsf{mac}_a(<\gamma,\tau>) \neq \delta$ $<\beta, pk_{\mathsf{svr}'}, \rho, \alpha> \leftarrow D_{sk_{\mathsf{svr}}}(\gamma)$ abort if $\beta \neq b$ $<u, d_{22}> \leftarrow D_{sk_{\mathsf{svr}}}(\zeta)$ abort if u is disabled $d_2 \leftarrow d_{12} + d_{22}$ $d'_{21} \leftarrow_R \{0,1\}^{\lambda+\kappa}$ $d'_{22} \leftarrow d_2 - d'_{21}$ $\zeta' \leftarrow E_{pk_{\mathsf{svr}'}}(<u, d'_{22}>)$ $\eta \leftarrow \rho \oplus <d'_{21}, \zeta'>$ $\delta' \leftarrow \mathsf{mac}_\alpha(\eta)$
	$\xleftarrow{\delta',\eta}$	
abort if $\mathsf{mac}_\alpha(\eta) \neq \delta'$ $<d'_{21}, \zeta'> \leftarrow \rho \oplus \eta$ $d'_{11} \leftarrow_R \{0,1\}^{\lambda+\kappa}$ $d'_{12} \leftarrow d_1 - d'_{11}$ $d'_1 \leftarrow d'_{11} + d'_{21}$ $v' \leftarrow_R \{0,1\}^\kappa$ $a' \leftarrow_R \{0,1\}^\kappa$ $b' \leftarrow f(v',\pi)$ $\tau' \leftarrow E_{pk_{\mathsf{svr}'}}(<a', b', d'_{12}, \zeta', N>)$ store $<\mathsf{svr}', pk_{\mathsf{svr}'}, \tau', t, v', d'_1, a'>$		

Figure 2: S-RSA-Del delegation protocol

new share d_2' for svr' is constructed as $d_2' = d_{12}' + d_{22}' = (d_1 - d_{11}') + (d_2 - d_{21}')$, with the first and second terms being computed by dvc and svr, respectively. As a result, $d_1' + d_2' = d_1 + d_2$. Note that the two terms d_{12}' and d_{22}' are actually stored separately in the ticket, with d_{22}' being encrypted by the consenting server so the device can not determine the full share d_2' when it constructs the new ticket.

In addition to the manipulation of these shares of d, this protocol borrows many components from the signature protocol of Figure 1. For example, β, γ and δ all play similar roles in the protocol as they did in Figure 1. And deletion is once again important: dvc must delete α, β, b', d_{11}', d_{21}', ρ, ζ', and all other intermediate computations at the completion of this protocol. Similarly, svr should delete α, β, b, d_{12}, d_{22}, d_{21}', d_{22}', ρ, ζ', and all other intermediate results when it completes.

The value α is important in this protocol for protecting against denial-of-service attacks by a network attacker. α is a mac key used to authenticate svr's response to dvc. This is important to prevent a network attacker from falsifying the value η to the dvc in the delegation protocol. Since dvc cannot verify the correctness of η, permitting the falsification of η could result in dvc completing this protocol with a corrupt and useless authorization record $<\mathsf{svr}', pk_{\mathsf{svr}'}, \tau', t, v', d_1', a'>$. Thus, dvc authenticates the response from svr using α to prevent a network attacker from falsifying η. It is important to note, however, that a corrupt svr could still undetectably generate η incorrectly, and so α provides limited (but practically important) protection in this regard. It is also interesting to contrast our use of α in this protocol with the absence of any such mechanism in the S-RSA-Del signature protocol in Figure 1: in Figure 1, dvc can detect an incorrect response, and so there we omit any further authentication of svr's response.

To relate this protocol to the system model of Section 2.1, and for our proofs in Section 5, we define the execution of the code before the first message in Figure 2 to constitute a $\mathsf{dvc.startDel}(\mathsf{svr}, \mathsf{svr}')$ event. Likewise, we define the execution of the code after the second message in Figure 2 to constitute a $\mathsf{dvc.finishDel}(\mathsf{svr}, \mathsf{svr}')$ event. The event $\mathsf{dvc.revoke}(\mathsf{svr})$, though not pictured in Figure 2, can simply be defined as dvc deleting any authorization record $<\mathsf{svr}, pk_{\mathsf{svr}}, \tau, t, v, d_1, a>$ and halting any ongoing delegation protocols to authorize svr.

3.5 Key disabling

As in [14], the S-RSA-Del system supports the ability to *disable* the device's key at servers, as would be appropriate to do if the device were stolen. Provided that the user backed up t before the device was stolen, the user can send t to a server svr. svr can then store $u = h_{\mathsf{dsbl}}(t)$ on a list of disabled ticket identifiers. Subsequently, svr should refuse to respond to any request containing a ticket τ such that $<*, *, *, \zeta, *> \leftarrow D_{sk_{\mathsf{svr}}}(\tau)$ and $<u, *> \leftarrow D_{sk_{\mathsf{svr}}}(\zeta)$. Rather than storing u forever, the server can discard u 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 attacker attempting to disable u without t), we do not need h_{dsbl} to be a random oracle, but simply a one-way hash function.

In relation to the model of Section 2.1, $\mathsf{svr.disable}$ denotes the event in which svr receives

t and marks $u = h_{\mathsf{dsbl}}(t)$ as disabled. For convenience, we say that a ticket τ is disabled at svr if $<*, *, *, \zeta, *> \leftarrow D_{sk_{\mathsf{svr}}}(\tau)$, $<u, *> \leftarrow D_{sk_{\mathsf{svr}}}(\zeta)$, and u is marked as disabled at svr.

4 Delegation for D-ElG

In addition to RSA signatures, ElGamal decryption [8] has also been used to demonstrate the general approach to achieving capture-resilient devices advocated here. Specifically, prior work [14] presented a protocol D-ElG by which the device can perform an ElGamal decryption only after validating the user's password at a designated remote server. In this section, we show a protocol achieving our goals, including delegation, for ElGamal decryption. We call this protocol D-ElG-Del.

4.1 Preliminaries

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 a 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 $y = g^x \bmod p$. Following [14], 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:

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

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-Del protocol is described in the following subsections.

There are several possibilities for ElGamal-like encryption schemes that, when used to instantiate the framework above, enable goals G1–G4 to be proved for our D-ElG-Del protocol. That said, the precise senses in which a particular instance can satisfy goal G4 deserve some discussion. The most natural definition of security for key disabling is that an adversary in class A4 who is presented with a ciphertext c *after* time $\max_{\mathsf{svr} \in U}\{T(\mathsf{svr.disable})\}$

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* this time, as long as c were not sent to any $\mathsf{svr} \in U$ before disabling.

If the original ElGamal scheme [8] is secure against indifferent chosen ciphertext attacks [19], then the D-ELG-DEL protocol 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 [17]). There are, however, ElGamal-like encryption schemes that suffice to achieve even the latter, stronger security property, such as the proposals in [21]. When our protocol is instantiated with one of these (as in [14]), D-ELG-DEL can be proved secure even in the stronger sense, in the random oracle model.

4.2 Device initialization

The inputs to device initialization are the identity of a server svr_0 along with its public encryption key pk_{svr_0}, 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:

$$
\begin{aligned}
& t \leftarrow_R \{0,1\}^\kappa \\
& u \leftarrow h_{\mathsf{dsbl}}(t) \\
& v \leftarrow_R \{0,1\}^\kappa \\
& a \leftarrow_R \{0,1\}^\kappa \\
& b \leftarrow f(v, \pi_0) \\
& x_0 \leftarrow h(\pi_0) \\
& x_1 \leftarrow_R \mathbb{Z}_q \\
& x_2 \leftarrow x - x_1 - x_0 \bmod q \\
& \zeta \leftarrow E_{pk_{\mathsf{svr}_0}}(<u, x_2>) \\
& \tau \leftarrow E_{pk_{\mathsf{svr}_0}}(<a, b, g, p, q, 0, \zeta>)
\end{aligned}
$$

We assume h produces an output of $\lambda + \kappa$ bits. The value pk_{dvc} and the authorization record $<\mathsf{svr}_0, pk_{\mathsf{svr}_0}, \tau, t, v, x_1, a>$, are saved on stable storage in the device. All other values, including b, u, x, x_0, x_2, ζ, and π_0, are deleted from the device. The value t 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.

4.3 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 the identity svr of the server to be used, such that dvc holds a record $<\mathsf{svr}, pk_{\mathsf{svr}}, \tau, t, v, x_1, a>$. Recall that dvc also stores $pk_{\mathsf{dvc}} = <g, p, q, y>$.

dvc		svr
abort if $\mathsf{valid}(c) = 0$ $\beta \leftarrow f(v, \pi)$ $\rho \leftarrow_R \{0,1\}^\lambda$ $\gamma \leftarrow E_{pk_{\mathsf{svr}}}(<c, \beta, \rho>)$ $\delta \leftarrow \mathsf{mac}_a(<\gamma, \tau>)$		
	$\xrightarrow{\gamma,\delta,\tau}$	
		$<a, b, g, p, q, x_{12}, \zeta> \leftarrow D_{sk_{\mathsf{svr}}}(\tau)$ abort if $\mathsf{mac}_a(\gamma, \tau) \neq \delta$ $<c, \beta, \rho> \leftarrow D_{sk_{\mathsf{svr}}}(\gamma)$ abort if $\beta \neq b \ \vee \ \mathsf{valid}(c) = 0$ $w \leftarrow \mathsf{select}(c)$ $<u, x_{22}> \leftarrow D_{sk_{\mathsf{svr}}}(\zeta)$ abort if u is disabled $x_2 \leftarrow x_{12} + x_{22} \bmod q$ $\nu \leftarrow w^{x_2} \bmod p$ $\eta \leftarrow \nu \oplus \rho$ $r \leftarrow_R \mathbb{Z}_q$ $e \leftarrow h_{\mathsf{zkp}}(<\nu, w^r \bmod p, g^r \bmod p>)$ $s \leftarrow x_2 e + r \bmod q$
	$\xleftarrow{\eta,e,s}$	
$\nu \leftarrow \rho \oplus \eta$ abort if $\nu^q \not\equiv_p 1$ $w \leftarrow \mathsf{select}(c)$ $x_0 \leftarrow h(\pi)$ abort if $e \neq h_{\mathsf{zkp}}(<\nu, w^s \nu^{-e} \bmod p,$ $g^s (y g^{-(x_0+x_1)})^{-e} \bmod p>)$ $\mu \leftarrow w^{x_0+x_1} \bmod p$ return $\mathsf{reveal}(\nu\mu \bmod p, c)$		

Figure 3: D-ElG-Del decryption protocol

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.

4.4 Delegation protocol

Here we present the protocol by which the device delegates the password checking function from an authorized server svr to another server svr′. The inputs provided to the device for this protocol are the identity of svr, such that dvc holds a record $<\mathsf{svr}, pk_{\mathsf{svr}}, \tau, t, v, x_1, a>$, the public key $pk_{\mathsf{svr}'}$ for svr′, and the input password π. (As stated above, one could also input policy information here.) Recall that dvc also stores $pk_{\mathsf{dvc}} = <g, p, q, y>$. The protocol is described in Figure 4.

The overall goal of the protocol in Figure 4 is to generate a new share x'_2 for server svr′, and new share x'_1 and new ticket τ' for the device to use with svr′. x'_1 and x'_2 are constructed in a fashion similar to how d'_1 and d'_2 were constructed in Figure 2, i.e., using correlated random offsets from x_1 and x_2 computed jointly by dvc and svr. In Figure 4, β, γ, δ, δ' and α play roles similar to those in S-RSA-Del. It is important that the device delete β, b', ρ and all other intermediate results when the protocol completes, and to never store them on stable storage.

4.5 Key disabling

Like S-RSA-Del, the D-ElG-Del protocol also supports key disabling. Assuming the user backed up t before the device was stolen, the user can send t to a server svr. svr then computes $u = h_{\mathsf{dsbl}}(t)$ and records u on a disabled list. Subsequently, svr should refuse to respond to any request containing a ticket τ such that $<*, *, *, *, *, *, \zeta> \leftarrow D_{sk_{\mathsf{svr}}}(\tau)$ and $<u, *> \leftarrow D_{sk_{\mathsf{svr}}}(\zeta)$. Rather than storing u forever, the server can discard u 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 u without t), we do not need h_{dsbl} to be a random oracle, but simply a one-way hash function.

5 Security for S-RSA-Del

In this section we provide a formal proof of security for the S-RSA-Del system in the random oracle model. We begin, however, with some intuition for the goals G1–G4 in light of the protocols of Figures 1 and 2.

- An A1 attacker never obtains an authorization record $<\mathsf{svr}, pk_{\mathsf{svr}}, \tau, t, v, d_1, a>$ from dvc, either because it never compromises dvc or because dvc has deleted all such records by

dvc		svr
$\beta \leftarrow f(v, \pi)$ $\rho \leftarrow_R \{0,1\}^{\kappa+\ell(\lambda)}$ $\alpha \leftarrow_R \{0,1\}^{\kappa}$ $\gamma \leftarrow E_{pk_{\mathsf{svr}}}(<\beta, pk_{\mathsf{svr}'}, \rho, \alpha>)$ $\delta \leftarrow \mathsf{mac}_a(<\gamma, \tau>)$		
	$\xrightarrow{\gamma,\delta,\tau}$	
		$<a, b, g, p, q, x_{12}, \zeta> \leftarrow D_{sk_{\mathsf{svr}}}(\tau)$ abort if $\mathsf{mac}_a(<\gamma, \tau>) \neq \delta$ $<\beta, pk_{\mathsf{svr}'}, \rho, \alpha> \leftarrow D_{sk_{\mathsf{svr}}}(\gamma)$ abort if $\beta \neq b$ $<u, x_{22}> \leftarrow D_{sk_{\mathsf{svr}}}(\zeta)$ abort if u is disabled $x_2 \leftarrow x_{12} + x_{22} \bmod q$ $x'_{21} \leftarrow_R \mathbb{Z}_q$ $x'_{22} \leftarrow x_2 - x'_{21} \bmod q$ $\zeta' \leftarrow E_{pk_{\mathsf{svr}'}}(<u, x'_{22}>)$ $\eta \leftarrow \rho \oplus <x'_{21}, \zeta'>$ $\delta' \leftarrow \mathsf{mac}_\alpha(\eta)$
	$\xleftarrow{\delta',\eta}$	
abort if $\mathsf{mac}_\alpha(\eta) \neq \delta'$ $<x'_{21}, \zeta'> \leftarrow \eta$ $x'_{11} \leftarrow_R \mathbb{Z}_q$ $x'_{12} \leftarrow x_1 - x'_{11} \bmod q$ $x'_1 \leftarrow x'_{11} + x'_{21} \bmod q$ $v' \leftarrow_R \{0,1\}^{\kappa}$ $a' \leftarrow_R \{0,1\}^{\kappa}$ $b' \leftarrow f(v', \pi)$ $\tau' \leftarrow E_{pk_{\mathsf{svr}'}}(<a', b', g, p, q, x'_{12}, \zeta'>)$ store $<\mathsf{svr}', pk_{\mathsf{svr}'}, \tau', t, v', x'_1, a'>$		

Figure 4: D-ElG-Del delegation protocol

the time it is compromised. Without any d_1 used with any svr, the attacker has no ability to forge a signature for dvc (even if it knows π_0 and thus d_0); this is property G1.

- An A2 attacker *can* obtain $<\mathsf{svr}, pk_{\mathsf{svr}}, \tau, t, v, d_1, a>$ for some svr, but only for a svr that is never compromised. Thus, the attacker has no information about the d_2 in τ (or more precisely, the values d_{12} and ζ encoded in τ, or the value d_{22} encoded in ζ) and can forge only by succeeding in an online dictionary attack with some such svr (goal G2).

- Now consider an A3 attacker. If some $\mathsf{svr} \in \mathsf{Active}(\mathsf{dvc.comp})$ is ever compromised, then the attacker can obtain its d_{12} and ζ by decrypting τ for svr, and then d_{22} by decrypting ζ. The A3 attacker can then conduct an offline dictionary attack on π_0 using the d_1 for svr and $d_2 \leftarrow d_{12} + d_{22}$, and so goal G3 is the best that can be achieved in this case.

- An attacker in class A4 compromises both π_0 (i.e., d_0) and dvc when there is at least one active svr (and so it learns d_1 for svr). Moreover, the attacker can delegate from svr to any other $\mathsf{svr}' \in U$, and obviously will learn the d_1' for that svr'. Thus, to achieve disabling, it is necessary that the attacker never corrupts any svr (and so never learns any d_2 for any svr). If this is the case, then goal G4 says that disabling all servers will prevent further forgeries.

We now proceed to a formal proof of goals G1–G4.

5.1 Definitions

To prove security of our system, we must first state requirements for the security of an encryption scheme, of a signature scheme, and of S-RSA-Del.

Security for encryption schemes We specify adaptive chosen-ciphertext security [19] for an encryption scheme $\mathcal{E} = (G_{enc}, E, D)$. (For more detail, see [1, 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 a scheme if the attacker makes q queries to the decryption oracle, and $\Pr(A \text{ succeeds}) \geq \frac{1}{2} + \epsilon$.

Security for signature schemes We specify existential unforgeability versus chosen message attacks [10] 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.

Security for S-RSA-DEL Let S-RSA-DEL$[\mathcal{E}, \mathcal{D}]$ denote an S-RSA-DEL system based on encryption scheme $\mathcal{E}$ and on dictionary $\mathcal{D}$. A forger is given $<e, N>$ where $(<e, N>, <d, N, \phi(N)>) \leftarrow G_{RSA}(1^\lambda)$, and the public data generated by the initialization procedure for the system. The initialization procedure specifies svr_0. The goal of the forger is to forge RSA signatures with respect to $<e, N>$. The forger is allowed to query a dvc oracle, a disable oracle, svr oracles, a password oracle, and (possibly) random oracles. A random oracle takes an input and returns a random hash of that input, in the defined range. A disable oracle query returns a value t that can be sent to the server to disable any ticket τ containing ζ having ticket identifier $u = h_{\mathsf{dsbl}}(t)$ at the server. A password oracle may be queried with comp, and returns π_0.

A svr oracle may be queried with handleSign, handleDel, disable, and comp. On a comp query, the svr oracle returns sk_{svr}. On a $\mathsf{handleSign}(\gamma, \delta, \tau)$ query, which represents the receipt of a message in the S-RSA-DEL signature protocol ostensibly from the device, it returns an output message η (with respect to the secret server data generated by the initialization procedure). On a $\mathsf{handleDel}(\gamma, \delta, \tau)$ query, which represents the receipt of a message in the S-RSA-DEL delegation protocol ostensibly from the device, it returns an output message δ', η. On a $\mathsf{disable}(t)$ query the svr oracle rejects all future queries with tickets containing ticket identifiers equal to $h_{\mathsf{dsbl}}(t)$ (see Section 3.5).

The dvc oracle may be queried with startSign, finishSign, startDel, finishDel, revoke, and comp. We assume there is an implicit notion of sessions so that the dvc oracle can determine the startSign query corresponding to a finishSign query and the startDel query corresponding to a finishDel query. On a $\mathsf{startSign}(m, \mathsf{svr})$ query, which represents a request to initiate the S-RSA-DEL signature protocol, if svr is authorized, the dvc oracle returns an output message γ, δ, τ, and sets some internal state (with respect to the secret device data and the password generated by the initialization procedure). On the corresponding $\mathsf{finishSign}(\eta)$ query, which represents the device's receipt of a response ostensibly from svr, the dvc oracle either aborts or returns a valid signature for the message m given as input to the previous startSign query. On a $\mathsf{startDel}(\mathsf{svr}, \mathsf{svr}')$ query, which represents a request to initiate the S-RSA-DEL delegation protocol, if svr is authorized, the dvc oracle returns an output message γ, δ, τ, and sets some internal state. On the corresponding $\mathsf{finishDel}(\delta', \eta)$ query, which represents the device's receipt of a response ostensibly from svr, the dvc oracle either aborts or authorizes svr', i.e., it creates a new authorization record for svr'. On a $\mathsf{revoke}(\mathsf{svr})$ query, the dvc oracle erases the authorization record for svr, thus revoking the authorization of svr. On a comp query, the dvc oracle returns all stored authorization records.

A class A1, A2, or A3 forger *succeeds* if after attacking the system it can output a pair $(m, <\sigma, r>)$ where $\sigma^e \equiv_N \mathsf{encode}(m, r)$ and there was no $\mathsf{startSign}(m, \mathsf{svr})$ query. A class A4 forger *succeeds* if after attacking the system it can output a pair $(m, <\sigma, r>)$ where $\sigma^e \equiv_N \mathsf{encode}(m, r)$ and there was no svr $\mathsf{handleSign}(\gamma, \delta, \tau)$ query, where $D_{sk_{\mathsf{svr}}}(\gamma) = <m, *, *, *>$, before all servers received $\mathsf{disable}(t)$ queries, where $u = h_{\mathsf{dsbl}}(t)$ is the ticket identifier generated in initialization.

Let q_{dvc} be the number of startSign and startDel queries to the device. Let q_{svr} be the number of handleSign and handleDel queries to the servers. For Theorem 5.2, 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-Del if it makes $|\overline{q}|$ oracle queries (of the respective type and to the respective oracles) and succeeds with probability at least ϵ.

5.2 Theorems

Here we prove that if a forger breaks the S-RSA-Del 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-Del 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.

We prove security separately for the different classes of attackers from Section 2.2. The idea behind each proof is a simulation argument. We assume that a forger F can break the S-RSA-Del system, and then depending on how F attacks the system, we show that we can use it to either break the underlying encryption scheme or break the underlying RSA signature scheme.

For security against all classes of forgers, we must assume h and f are random oracles, and that f_v (for random v) and h have negligible probabilities of collision over $\mathcal{D}$. (The latter property would be achieved, for example, if h and f were random oracles and $|\mathcal{D}| < 2^{c\kappa}$ for a constant $c < \frac{1}{2}$.) However, for certain types of forgers, weaker hash function properties suffice. For proving security against a forger in class A2, we make no requirement on h, and we only require $\{f_v\}_v$ to be a pseudorandom function family, and that f_v (for random v) has a negligible probability of collision over $\mathcal{D}$. For proving security against a class A1 or class A4 forger we make no requirement on h or f.

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 5.1 *If a class A1 forger $(\overline{q}, \epsilon)$-breaks the* S-RSA-Del$[\mathcal{E}, \mathcal{D}]$ *system, then there is a forger that $(q_{\mathsf{dvc}}, \epsilon')$-breaks the RSA signature scheme with $\epsilon' \approx \epsilon$.*

Proof: Assume a class A1 forger F forges in the S-RSA-Del system with probability ϵ. Then we show how to construct a forger F^* for the underlying RSA signature scheme that succeeds with probability $\epsilon' \approx \epsilon$. Suppose F^* is given an RSA public key $<e, N>$ and a corresponding signature oracle.

F^* runs the following simulation of the S-RSA-Del system. F^* sets the dvc public key to $<e, N>$, and chooses $\pi_0 \leftarrow_R \mathcal{D}$. It generates all server key pairs $\{(pk_{\mathsf{svr}}, sk_{\mathsf{svr}})\}_{\mathsf{svr} \in U}$, and gives $\{pk_{\mathsf{svr}}\}_{\mathsf{svr} \in U}$ to F. F^* then constructs the authorization record $<\mathsf{svr}_0, pk_{\mathsf{svr}_0}, \tau, t, v, d_1, a>$ as normal, except setting $d_2 \leftarrow_R \mathbb{Z}_N$. F^* stores this authorization record and this corresponding value of d_2; d_2 is called the "correct server share" for τ. F^* then responds to oracle queries from F as follows:

- F^* responds to svr and disable oracle queries as in the real system.
- F^* responds to a $\mathsf{startSign}(m, \mathsf{svr})$ query by querying the signature oracle to get σ and r (or deriving r from σ, if necessary), and responding as in the real protocol using that r value. F^* responds to a $\mathsf{finishSign}(\eta)$ query corresponding to a $\mathsf{startSign}(m, \mathsf{svr})$ query by looking up the authorization record $<\mathsf{svr}, pk_{\mathsf{svr}}, \tau, t, v, d_1, a>$ and the correct server share d_2 for τ, computing $\nu \leftarrow \eta \oplus \rho$ (for the ρ value from the $\mathsf{startSign}(m, \mathsf{svr})$ query) and checking that $\nu \equiv_N (\mathsf{encode}(m, r))^{d_2}$. If this is false, F^* simulates a dvc abort. Otherwise, F^* returns $<\sigma, r>$.
- F^* responds to $\mathsf{startDel}(\mathsf{svr}, \mathsf{svr}')$ and revoke queries as in the real system. F^* responds to a $\mathsf{finishDel}(\delta', \eta)$ query corresponding to a $\mathsf{startDel}(\mathsf{svr}, \mathsf{svr}')$ query by looking up the authorization record $<\mathsf{svr}, pk_{\mathsf{svr}}, \tau, t, v, d_1, a>$ and the correct server share d_2 for τ, behaving as in the real protocol to generate and store a new authorization record $<\mathsf{svr}', pk_{\mathsf{svr}'}, \tau', t, v', d_1', a'>$, and storing $d_2' \leftarrow d_2 - d_{21}' + d_{12}'$ as the correct server share for τ'.

From F's perspective, the above simulation is statistically indistinguishable from S-RSA-Del. (It is not perfectly indistinguishable from S-RSA-Del due to the choice of d_2 in initialization.) So, if F forges with probility ϵ in S-RSA-Del, then it forges with probability $\epsilon' \approx \epsilon$ in this simulation. To break the RSA signature scheme with probability $\epsilon' \approx \epsilon$, F^* simply runs F in this simulation and outputs any forgery produced by F. □

Theorem 5.2 *Let h and f be random oracles. Also suppose that f_v (for random v) and h have negligible probabilities of collision over $\mathcal{D}$. If a class A3 forger $(\overline{q}, \epsilon)$-breaks the* S-RSA-Del$[\mathcal{E}, \mathcal{D}]$ *system with $\epsilon = \frac{q_h + q_f}{|\mathcal{D}|} + \psi$, then there is a forger F^* that $(q_{\mathsf{dvc}}, \epsilon')$-breaks the RSA signature scheme with $\epsilon' \approx \psi$.*

Proof: Assume a class A3 forger F forges in the S-RSA-Del system with probability $\frac{q_h + q_f}{|\mathcal{D}|} + \psi$. Then we show how to break the underlying RSA signature scheme with probability $\epsilon' \approx \psi$. Say we are given an RSA public key $<e, N>$ and a corresponding signature oracle. We construct a simulation of the S-RSA-Del system as in the proof of Theorem 5.1, except that the simulation aborts if the h oracle or f oracle is queried with π_0. This simulation is statistically indistinguishable from the real system (from F's viewpoint) unless the simulation aborts, the probability of which is exactly that of an offline dictionary attack (i.e., bounded by at most $\frac{q_h + q_f}{|\mathcal{D}|}$ plus the probability of a collision of one of the random oracles over $\mathcal{D}$). So since F forges with probability $\frac{q_h+q_f}{|\mathcal{D}|}+\psi$ in the real system, it forges with probability at least $\epsilon' \approx \psi$ in the simulation. Then to break the RSA signature scheme, we simply run F in this simulation and output any forgery produced by F. ∎

Theorem 5.3 *Suppose that $\{f_v\}_{v\in\{0,1\}^\kappa}$ is a pseudorandom function family, and that f_v (for random v) has a negligible probability of collision over $\mathcal{D}$. If a class A2 forger $(\overline{q},\epsilon)$-breaks the* S-RSA-DEL$[\mathcal{E},\mathcal{D}]$ *system where $\epsilon = \frac{q_{\mathsf{svr}}}{|\mathcal{D}|}+\psi$, then either there is an attacker A that $(3q_{\mathsf{svr}},\epsilon'')$-breaks $\mathcal{E}$ where $\epsilon'' \approx \frac{\psi}{2|U|(2q_{\mathsf{dvc}}+1)}$, or there is a forger F^* that $(q_{\mathsf{dvc}},\epsilon')$-breaks the RSA signature scheme with $\epsilon' \approx \frac{\psi}{2}$.*

Proof: Assume a class A2 forger F forges in the S-RSA-DEL system with probability $\frac{q_{\mathsf{svr}}}{|\mathcal{D}|}+\psi$. Now define a server to be *good* if F never compromises it. (Recall that we consider static adversaries only, and so the good servers are determined before the system begins.) Let Real' be the S-RSA-DEL system in which all instances of f_v used for servers that are not good are replaced by perfectly random functions, and say F forges in the Real' system with probability $\frac{q_{\mathsf{svr}}}{|\mathcal{D}|}+\psi'$. Then by the pseudorandomness of $\{f_v\}_{v\in\{0,1\}^\kappa}$, $\psi' \approx \psi$. Now we construct a simulator Sim for Real' that takes an RSA public key $<e,N>$ and corresponding signature oracle as input. If forger F *wins* (as defined below) against Sim with probability greater than $\frac{q_{\mathsf{svr}}}{|\mathcal{D}|}+\frac{\psi'}{2}$, then we will be able to construct a forger that $(q_{\mathsf{dvc}},\epsilon')$-breaks the RSA signature scheme with $\epsilon' \approx \frac{\psi'}{2}$. If, on the other hand, F wins against Sim with probability at most $\frac{q_{\mathsf{svr}}}{|\mathcal{D}|}+\frac{\psi'}{2}$, then we will construct an attacker that $(3q_{\mathsf{svr}},\epsilon'')$-breaks $\mathcal{E}$ with $\epsilon'' \approx \frac{\psi}{2|U|(2q_{\mathsf{dvc}}+1)}$. We say that F *wins* against Sim if either F produces a valid forgery under $<e,N>$ or F makes a *successful online password guess*. The latter happens if F makes a query to a good svr with input (γ,δ,τ), such that if $<a,*,*,*,*> \leftarrow D_{sk_{\mathsf{svr}}}(\tau)$ then $\delta = \mathsf{mac}_a(<\gamma,\tau>)$, and either

- τ is a ticket stored on the device for svr, v is stored in the authorization record with τ, and either
  - for a $\mathsf{handleSign}$ query to svr, γ is not generated by a device $\mathsf{startSign}(*,\mathsf{svr})$ query, $<*,*,\beta,*> \leftarrow D_{sk_{\mathsf{svr}}}(\gamma)$, and $\beta = f(v,\pi_0)$,
  - for a $\mathsf{handleDel}$ query to svr, γ is not generated by a device $\mathsf{startDel}(\mathsf{svr},*)$ query, $<\beta,*,*,*> \leftarrow D_{sk_{\mathsf{svr}}}(\gamma)$, and $\beta = f(v,\pi_0)$; or
- τ is not a ticket stored on the device for svr, γ was generated by a device $\mathsf{startSign}(*,\mathsf{svr})$ or $\mathsf{startDel}(\mathsf{svr},*)$ query using a record $<\mathsf{svr},*,*,*,v,*,*>$, $<*,b',*,*,*> \leftarrow D_{sk_{\mathsf{svr}}}(\tau)$, and $b' = f(v,\pi_0)$.

We now define Sim. Below, we say that Sim *zeroes* a ticket τ for svr if Sim generates $\tau \leftarrow E_{pk_{\mathsf{svr}}}(0^{3\kappa+\ell(\lambda)+2\lambda})$, and we call the values $<a,b,d_{12},\zeta,N>$ present when Sim creates τ the *zeroed inputs* to τ. Similarly, Sim zeroes a value γ for svr if Sim generates $\gamma \leftarrow E_{pk_{\mathsf{svr}}}(0^{|m|+\kappa_{sig}+\kappa+\lambda})$ (respectively, $\gamma \leftarrow E_{pk_{\mathsf{svr}}}(0^{3\kappa+\lambda+\ell(\lambda)}))$ in a $\mathsf{startSign}(m,\mathsf{svr})$ (resp.,

$\mathsf{startDel}(\mathsf{svr}, \mathsf{svr}'))$ oracle query, and the values $<m, r, \beta, \rho>$ (resp., $<\beta, pk_{\mathsf{svr}'}, \rho, \alpha>$) are its *zeroed inputs*, respectively.

Sim gives $<e, N>$ to F as the device's public signature key. Then Sim generates all servers' key pairs $\{(pk_{\mathsf{svr}}, sk_{\mathsf{svr}})\}_{\mathsf{svr}\in U}$, and gives $\{pk_{\mathsf{svr}}\}_{\mathsf{svr}\in U}$ to F. Next Sim generates $\pi_0 \leftarrow_R \mathcal{D}$ and the data $<a, b, 0, \zeta, N>$ for the ticket τ in the normal way, except that d_2 is chosen as $d_2 \leftarrow_R \mathbb{Z}_N$. If svr_0 is good, then Sim zeroes τ for svr_0, and else Sim sets $\tau \leftarrow E_{pk_{\mathsf{svr}_0}}(<a, b, 0, \zeta, N>)$. d_2 is called the "correct server share" for τ.

Sim responds to oracle queries as follows (using a truly random function for f, as in Real$'$).

- Sim responds to a dvc.comp query by giving all authorization records stored on the device to F. Sim responds to a svr.comp query by giving sk_{svr} to F.
- Sim responds to a svr $\mathsf{disable}(t')$ query by storing $u' = h_{\mathsf{dsbl}}(t')$. Subsequently, any ticket τ of the following form is considered disabled at svr: either τ is zeroed for svr with zeroed inputs $<*, *, *, \zeta, *>$ or $<*, *, *, \zeta, *> \leftarrow D_{sk_{\mathsf{svr}}}(\tau)$; and $<u', *> \leftarrow D_{sk_{\mathsf{svr}}}(\zeta)$.
- Sim responds to a svr $\mathsf{handleSign}(\gamma, \delta, \tau)$ or $\mathsf{handleDel}(\gamma, \delta, \tau)$ query for a τ that has not been disabled at svr as a normal server would, except for the following changes:
  - If τ or γ was zeroed for svr, then its zeroed inputs are used in the handleSign or handleDel processing. Otherwise, their actual decryptions using sk_{svr} are used.
  - Sim aborts in the event of a successful password guess.
- Sim responds to a dvc $\mathsf{startSign}(m, \mathsf{svr})$ query as in Theorem 5.1, except if svr is good, it zeroes γ. Sim responds to a dvc $\mathsf{finishSign}(\eta)$ query as in Theorem 5.1.
- Sim responds to a dvc $\mathsf{startDel}(\mathsf{svr}, \mathsf{svr}')$ query as in Theorem 5.1, except if svr is good, it zeroes γ. (Recall that there is no startDel query after dvc.comp, by our assumption that compromising the device implies it is captured.) Sim responds to a dvc $\mathsf{finishDel}(\delta', \eta)$ query corresponding to a $\mathsf{startDel}(\mathsf{svr}, \mathsf{svr}')$ query as in Theorem 5.1, except if svr' is good, it zeroes τ'.

Suppose that the probability of F winning against Sim is more than $\frac{q_{\mathsf{svr}}}{|\mathcal{D}|} + \frac{\psi}{2}$. Since all instances of f_v for non-good servers are replaced by random functions, and all τ or γ ciphertexts encrypted under the public keys of good servers are zeroed, F obtains no information on the password from Sim, and thus the probability of F making a successful online password guess is at most $\frac{q_{\mathsf{svr}}}{|\mathcal{D}|}$ plus the probability of a collision in f_v over $\mathcal{D}$ for one of q_{dvc} random v's. So F forges in Sim with probability at least $\epsilon' \approx \frac{\psi}{2}$. A forger F^* for the RSA signature scheme can thus run Sim for F and output any forgery by F to $(q_{\mathsf{dvc}}, \epsilon')$-break the RSA signature scheme.

Now assume that the probability of F winning against Sim is at most $\frac{q_{\mathsf{svr}}}{|\mathcal{D}|} + \frac{\psi}{2}$. Since F forges in Real$'$ with probability $\frac{q_{\mathsf{svr}}}{|\mathcal{D}|} + \psi'$, then it wins in the Real$'$ with at least that probability. Then we construct an attacker A that breaks $\mathcal{E}$ with probability $\epsilon'' \approx \frac{\psi}{2|U|(2q_{\mathsf{dvc}}+1)}$. Our attacker A is given a public key pk from $\mathcal{E}$ and corresponding decryption oracle, and runs a simulation of the Real' system for F, using a device signing key $<e, N>$ and private key $<d, N, \phi(N)>$ that it generates itself.

First consider a simulator that gives pk to F as the public key pk_{svr} of some svr that is good, and then simulates Real' exactly, except for aborting on a successful password guess and using a decryption oracle to decrypt messages encrypted under key pk by the adversary. There will be at most $3q_{\mathsf{svr}}$ 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 Real' to F (at least until F wins). Now consider the same simulation, but with all τ and γ values for good servers generated by the device zeroed. (Naturally, the server pretends the encryptions are of the normal messages, not strings of zeros.) The latter simulation is statistically indistinguishable from Sim. Thus, the probability of F winning in the latter simulation is at most $\frac{q_{\mathsf{svr}}}{|\mathcal{D}|} + \frac{\psi}{2}$ plus a negligible term due to the fact that the latter simulation is not perfectly indistinguishable from Sim, while the probability of F winning 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 $j \in \{0, \ldots, 2q_{\mathsf{dvc}}+1\}$ correspond to the first j τ-ciphertexts or γ-ciphertexts (generated by A) to good servers being of the normal messages (and all ζ ciphertexts being of the normal messages), and the remainder being encryptions of strings of 0's, and let p_j be the probability of F winning in experiment j. Then the average value for $i \in \{0, \ldots, 2q_{\mathsf{dvc}}\}$ of $p_{i+1} - p_i$ is at least $\approx \frac{\psi}{2(2q_{\mathsf{dvc}}+1)}$. Therefore, to construct A, we simply have A choose a random value $i \in \{0, \ldots, 2q_{\mathsf{dvc}} + 1\}$, assign $pk_{\mathsf{svr}} \leftarrow pk$ for a random good server svr, and run experiment i as above, but if the $(i+1)^{\text{st}}$ encryption to be generated by the simulator is to use pk_{svr}, it calls the test oracle for this encryption, 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 wins (meaning it believes X_0 was encrypted by the test oracle), and 1 otherwise. By the analysis above, A^* breaks $\mathcal{E}$ with probability $\epsilon'' \approx \frac{\psi}{2|U|(2q_{\mathsf{dvc}}+1)}$. □

Theorem 5.4 *Suppose the RSA signature scheme is deterministic (i.e., $\kappa_{sig} = 0$). If a class A4 forger $(\overline{q}, \epsilon)$-breaks the* S-RSA-DEL$[\mathcal{E}, \mathcal{D}]$ *system, then there is either an attacker A that $(3q_{\mathsf{svr}}, \frac{\epsilon}{2|U|(q_{\mathsf{dvc}}+1)})$-breaks $\mathcal{E}$ or a forger F^* that $(q_{\mathsf{svr}}, \frac{\epsilon}{2})$-breaks the RSA signature scheme.*

Proof: Assume a class A4 forger F forges in the S-RSA-DEL system with probability ϵ. Consider the following simulation Sim of S-RSA-DEL for F. Sim is given an RSA public key $<e, N>$ and a corresponding signature oracle. Sim sets the dvc public key to $<e, N>$, and chooses $\pi_0 \leftarrow_R \mathcal{D}$. It generates all server key pairs $\{(pk_{\mathsf{svr}}, sk_{\mathsf{svr}})\}_{\mathsf{svr} \in U}$, and gives $\{pk_{\mathsf{svr}}\}_{\mathsf{svr} \in U}$ to F. Sim then constructs τ as in the real system, except with $d_2 \leftarrow 0$ and $\zeta \leftarrow E_{pk_{\mathsf{svr}_0}}(0^{\lambda+2\kappa})$. Sim then saves the authorization record $<\mathsf{svr}_0, pk_{\mathsf{svr}_0}, \tau, t, v, d_1, a>$. For the ciphertext ζ, the tuple $<pk_{\mathsf{svr}_0}, \zeta, d_1>$ is recorded on an *offset list.*[2] Sim then responds to oracle queries from F as follows:

[2]The third value in this tuple is an overestimate of the value of the device share d_1, including the value d_{12} that is added to create the server share d_2. For the original ticket τ, this d_{12} value is actually zero, and thus the third value in the tuple is the true value of the device share d_1.

- Sim responds to dvc oracle queries as in the real system.
- Sim responds to a svr handleDel(γ, δ, τ) query as svr would normally, except for the following changes. Sim runs normally until either the decryption of ζ or until svr aborts. If svr does not abort, then Sim has computed $<a, b, d_{12}, \zeta, N> \leftarrow D_{sk_{\mathsf{svr}}}(\tau)$ and $<\beta, pk_{\mathsf{svr}'}, \rho, \alpha> \leftarrow D_{sk_{\mathsf{svr}}}(\gamma)$. Now Sim examines the offset list with pk_{svr} and ζ. If for some d_1^+, $<pk_{\mathsf{svr}}, \zeta, d_1^+>$ appears in the offset list, then Sim causes svr to abort if $u = h_{\mathsf{dsbl}}(t)$ is marked as disabled at svr, and otherwise
  - computes $\zeta' \leftarrow E_{pk_{\mathsf{svr}}}(0^{\lambda+2\kappa})$,
  - computes $d_1 \leftarrow d_1^+ - d_{12}$,
  - computes $d'_{21} \leftarrow \{0,1\}^{\lambda+\kappa}$ (i.e., as normal), and
  - stores $<pk_{\mathsf{svr}'}, \zeta', d_1 + d'_{21}>$ on the offset list.

  Sim then completes the computation of η and δ' and responds. If $<pk_{\mathsf{svr}}, \zeta, *>$ does not appear on the offset list, then Sim responds as svr would normally.
- Sim responds to a svr handleSign(γ, δ, τ) query as svr would normally, except for the following changes: Sim runs normally until either the decryption of ζ or until svr aborts. If svr does not abort, then Sim has computed $<a, b, d_{12}, \zeta, N> \leftarrow D_{sk_{\mathsf{svr}}}(\tau)$ and $<m, r, \beta, \rho> \leftarrow D_{sk_{\mathsf{svr}}}(\gamma)$. Now Sim examines the offset list with pk_{svr} and ζ. If for some d_1^+, $<pk_{\mathsf{svr}}, \zeta, d_1^+>$ appears in the offset list, then Sim causes svr to abort if u is marked as disabled at svr, and otherwise
  - computes $d_1 \leftarrow d_1^+ - d_{12}$,
  - invokes the signature oracle with the message m to be signed to obtain signature $<\sigma, r'>$ (though note that $|r'| = \kappa_{sig} = 0$ by assumption), and
  - computes $\nu \leftarrow \sigma(\mathsf{encode}(m, r')^{-d_0-d_1}) \bmod N$.

  Sim then completes the computation of η and responds. If $<pk_{\mathsf{svr}}, \zeta, *>$ does not appear on the offset list, then Sim responds as svr would normally.

  We note that once u is marked as disabled at all servers, there will never be a need to call the signature oracle again.
- Sim responds to a svr disable(t') query by marking $u' = h_{\mathsf{dsbl}}(t')$ as disabled at svr.
- Sim responds to a dvc comp query by returning the authorization records on dvc. Note that there are no svr comp queries, because the attacker is in class A4.

If F forges with probability greater than $\frac{\epsilon}{2}$ in Sim, then there is a forger F^* for the underlying RSA signature scheme that forges with probability $\frac{\epsilon}{2}$. F^* is given an RSA public key $<e, N>$ and a corresponding signature oracle. F^* runs Sim for F and outputs any forgery produced by F.

Now suppose that F wins in Sim with probability at most $\frac{\epsilon}{2}$. We use a hybrid argument to construct an attacker A that breaks $\mathcal{E}$ with probability at least $\frac{\epsilon}{2|U|(q_{\mathsf{svr}}+1)}$, as follows. A is given a public key pk from $\mathcal{E}$ with corresponding decryption and test oracles, and sets $pk_{\mathsf{svr}^*} \leftarrow pk$ for a randomly chosen svr^*. In addition, A generates $(<e, N>, <d, N, \phi(N)>) \leftarrow G_{RSA}(1^\lambda)$ and sets the device's public signing key to $<e, N>$. A then chooses an index $i \leftarrow_R \{0, \ldots, q_{\mathsf{svr}}\}$. A computes d_0, d_1, d_2, t, u, v, a, and b normally, but sets ζ as follows: if $i = 0$ and $\mathsf{svr}_0 = \mathsf{svr}^*$, then it sets ζ to be the output of the test oracle for pk queried with $X_0 = <u, d_2>$ and $X_1 = 0^{\kappa+\lambda}$ and records the tuple $<pk_{\mathsf{svr}_0}, \zeta, d_2>$ on an *offset list*.[3] If $i > 0$ or $\mathsf{svr}_0 \neq \mathsf{svr}^*$, then it sets ζ normally and records the tuple $<pk_{\mathsf{svr}_0}, \zeta, d_2>$ on the offset list. After this, A proceeds normally to complete the authorization record $<\mathsf{svr}_0, pk_{\mathsf{svr}_0}, \tau, t, v, d_1, a>$. A then simulates S-RSA-Del for F as follows.

- A responds to dvc oracle queries as in the real system.

- A responds to a svr $\mathsf{handleDel}(\gamma, \delta, \tau)$ query as follows. If either γ or τ are the output of the test oracle (if the test oracle has been queried), then svr aborts. Otherwise, A runs normally—using the decryption oracle for pk to decrypt γ and τ if $\mathsf{svr} = \mathsf{svr}^*$—until either the decryption of ζ or until svr aborts. If svr does not abort, suppose this is the j-th $\mathsf{handleDel}$ query (to any server) in which the server did not abort. At this point A has computed $<a, b, d_{12}, \zeta, N> \leftarrow D_{sk_{\mathsf{svr}}}(\tau)$ and $<\beta, pk_{\mathsf{svr}'}, \rho, \alpha> \leftarrow D_{sk_{\mathsf{svr}}}(\gamma)$ (or computed them using the decryption oracle). Now A examines the offset list with pk_{svr} and ζ:

  - If $<pk_{\mathsf{svr}}, \zeta, *>$ does not appear in the offset list, then if $pk \neq pk_{\mathsf{svr}}$, A decrypts ζ to obtain $<u', d_{22}>$, and if $pk = pk_{\mathsf{svr}}$ then A calls the decryption oracle on ζ to obtain $<u', d_{22}>$. If u' is marked as disabled at svr, then A simulates a svr abort, and otherwise A continues as svr would normally.
  - If for some d_{22}, $<pk_{\mathsf{svr}}, \zeta, d_{22}>$ appears in the offset list, then A has svr abort if $u = h_{\mathsf{dsbl}}(t)$ is marked as disabled at svr, and otherwise proceeds as follows: A sets $d_2 \leftarrow d_{12} + d_{22}$, computes d'_{21} and d'_{22} as normal, and alters the computation of ζ' as follows. If $j < i$, then A computes $\zeta' \leftarrow E_{pk_{\mathsf{svr}'}}(<u, d'_{22}>)$ as normal and records $<pk_{\mathsf{svr}'}, \zeta', d'_{22}>$ in the offset list. If $j = i$ and $\mathsf{svr}' = \mathsf{svr}^*$, then A calls the test oracle for pk with $X_0 = <u, d'_{22}>$ and $X_1 = 0^{\lambda+2\kappa}$, obtains the response ζ', and records $<pk_{\mathsf{svr}'}, \zeta', d'_{22}>$ in the offset list. If $j > i$, or $j = i$ and $\mathsf{svr}' \neq \mathsf{svr}^*$, then A sets $\zeta' \leftarrow E_{pk_{\mathsf{svr}'}}(0^{\lambda+2\kappa})$ and records $<pk_{\mathsf{svr}'}, \zeta', d'_{22}>$ in the offset list. In all cases, A then computes η and δ' normally and responds.

- A responds to a svr $\mathsf{handleSign}(\gamma, \delta, \tau)$ query as follows. If either γ or τ are the output of the test oracle (if the test oracle has been queried), then svr aborts. Otherwise, A runs normally—using the decryption oracle for pk to decrypt γ and τ if $\mathsf{svr} = \mathsf{svr}^*$—until either the decryption of ζ or until svr aborts. If svr does not abort, then Sim

[3] The third value in the tuple is the value that is supposed to be encrypted in ζ (but may not be the actual value that is encrypted).

has computed $<a, b, d_{12}, \zeta, N> \leftarrow D_{sk_{\mathsf{svr}}}(\tau)$ and $<m, r, \beta, \rho> \leftarrow D_{sk_{\mathsf{svr}}}(\gamma)$ (or computed them using the decryption oracle). Now A examines the offset list with pk_{svr} and ζ:

- If $<pk_{\mathsf{svr}}, \zeta, *>$ does not appear in the offset list, then if $pk \neq pk_{\mathsf{svr}}$, A decrypts ζ to obtain $<u', d_{22}>$, and if $pk = pk_{\mathsf{svr}}$ then A calls the decryption oracle on ζ to obtain $<u', d_{22}>$. If u' is marked as disabled at svr, then A simulates a svr abort, and otherwise A continues as svr would normally.
- If for some d_{22}, $<pk_{\mathsf{svr}}, \zeta, d_{22}>$ appears in the offset list, then A has svr abort if $u = h_{\mathsf{dsbl}}(t)$ is marked as disabled at svr. If u is not marked as disabled, then A sets $d_2 \leftarrow d_{12} + d_{22}$, sets $\nu \leftarrow \mathsf{encode}(m, r)^{d_2} \bmod N$, and responds as svr would normally.

- A responds to a svr disable(t') query by marking $u' = h_{\mathsf{dsbl}}(t')$ as disabled at svr.
- A responds to a dvc comp query by returning the authorization records on dvc. Note that there are no svr comp queries, because the attacker is in class A4.

Finally, A outputs 0 if F forges (meaning it believes X_0 was encrypted by the test oracle), and 1 otherwise.

Let experiment $j \in \{0, \ldots, q_{\mathsf{svr}} + 1\}$ correspond to the first j ζ-ciphertexts (generated by A), i.e., the ciphertexts internal to the tickets, being of normal messages, and the remainder being encryptions of zero. Let p_j be the probability of F winning in experiment j. Note that experiment 0 is perfectly indistinguishable from Sim, and experiment $q_{\mathsf{svr}} + 1$ is perfectly indistinguishable from the real system. Therefore $p_0 \leq \frac{\epsilon}{2}$ and $p_{q_{\mathsf{svr}}+1} \geq \epsilon$. Note that A will run either experiment i or $i + 1$, depending on the output of the test oracle, and if svr* is the server that receives encryption request $i + 1$. Then the average value for $i \in \{0, \ldots, q_{\mathsf{svr}}\}$ of $p_{i+1} - p_i$ is at least $\approx \frac{\epsilon}{2(q_{\mathsf{svr}}+1)}$, and the probability with which A breaks $\mathcal{E}$ is at least $\frac{\epsilon}{2|U|(q_{\mathsf{svr}}+1)}$. $\square$

6 Security for D-ElG-Del

In this section we provide a formal proof of security for the D-ElG-Del system instantiated with an ElGamal-like encryption scheme. Specifically, we provide a formal proof of goals G1–G4.

6.1 Definitions

We first define security for an ElGamal-like (ElGL) encryption scheme, and for the D-ElG-Del 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 $\mathsf{reveal}(z,c)$.

Note: The TDH1 and TDH2 encryption schemes from [21], when restricted to a single server, are in fact ElGL encryption schemes secure in the this sense (i.e., secure against adaptive chosen ciphertext attacks with a decryption oracle that returns $z = (\mathsf{select}(c))^x \bmod p$ instead of the decryption of c).

Security for D-ElG-Del Let D-ElG-Del$[\mathcal{E},\mathcal{D}]$ denote an D-ElG-Del system based on an encryption scheme $\mathcal{E}$ 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 system. The initialization procedure specifies svr_0. The attacker is allowed to query a dvc oracle, a disable oracle, svr oracles, a password oracle, a test oracle, and (possibly) random oracles. A random oracle takes an input and returns a random hash of that input, in the defined range. A disable oracle query returns a value t that can be sent to a server to disable any ticket τ containing ζ having ticket identifier $u = h_{\mathsf{dsbl}}(t)$ at the server. A password oracle may be queried with comp, and returns π_0.

A svr oracle may be queried with handleDecr, handleDel, disable, and comp. On a comp query, the svr oracle returns sk_{svr}. On a $\mathsf{handleDecr}(\gamma,\delta,\tau)$ query, which represents the receipt of a message in the D-ElG-Del decryption protocol ostensibly from the device, it returns an output message η, e, s (with respect to the secret server data generated by the initialization procedure). On a $\mathsf{handleDel}(\gamma,\delta,\tau)$ query, which represents the receipt of a message in the D-ElG-Del delegation protocol ostensibly from the device, it returns an output message δ', η. After a $\mathsf{disable}(t)$ query the svr oracle rejects all future queries with tickets containing (ζ-values containing) ticket identifiers equal to $h_{\mathsf{dsbl}}(t)$ (see Section 4.5).

The dvc oracle may be queried with startDecr, finishDecr, startDel, finishDel, revoke, and comp. We assume there is an implicit notion of sessions so that the dvc oracle can determine the startDecr query corresponding to a finishDecr query and the startDel query corresponding to a finishDel query. On a $\mathsf{startDecr}(c,\mathsf{svr})$ query, which represents a request to initiate the D-ElG-Del decryption protocol, if svr is authorized, the dvc oracle returns an output message γ,δ,τ, and sets some internal state (with respect to the secret device data and the password generated by the initialization procedure). On the corresponding $\mathsf{finishDecr}(\eta,e,s)$

query, which represents the device's receipt of a response ostensibly from svr, the dvc oracle either aborts or returns a valid decryption for the ciphertext c given as input to the previous $\mathsf{startDecr}$ query. On a $\mathsf{startDel}(\mathsf{svr}, \mathsf{svr}')$ query, which represents a request to initiate the D-ELG-DEL delegation protocol, if svr is authorized, the dvc oracle returns an output message γ, δ, τ, and sets some internal state. On the corresponding $\mathsf{finishDel}(\delta', \eta)$ query, which represents the device's receipt of a response ostensibly from svr, the dvc oracle either aborts or authorizes svr', i.e., it creates a new authorization record for svr'. On a $\mathsf{revoke}(\mathsf{svr})$ query, the dvc oracle erases the authorization record for svr, thus revoking the authorization of svr. On a comp query, the dvc oracle returns all stored authorization records.

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 class A1, A3, or A2, then it cannot query the dvc with a $\mathsf{startDecr}(Y, \mathsf{svr})$ query, and if A is of class A4, then it cannot query the svr with $\mathsf{handleDecr}(\gamma, \delta, \tau)$ query, where $D_{sk_{\mathsf{svr}}}(\gamma) = <Y, *, *>$, before all servers received $\mathsf{disable}(t)$ queries, where $u = h_{\mathsf{dsbl}}(t)$ is the ticket identifier generated in initialization. Finally A outputs b', and succeeds if $b' = b$.

Let q_{dvc}, q_{svr}, q_h, q_f be defined in the same manner as in Section 5. 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 that A $(\overline{q}, \epsilon)$-breaks D-ELG-DEL if A makes $|\overline{q}|$ oracle queries (of the respective types and to the respective oracles), and $\Pr(A \text{ succeeds}) \geq \frac{1}{2} + \epsilon$.

6.2 Theorems

Here we prove that if an attacker breaks the D-ELG-DEL$[\mathcal{E}, \mathcal{D}]$ 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 encryption scheme $\mathcal{E}$ can be broken with non-negligible probability. This implies that if the underlying ELGL encryption scheme and $\mathcal{E}$ are secure, our system will be as secure as inherently possible.

As in Section 5, we prove security separately for the different classes of attackers from Section 2.2.

Theorem 6.1 *Let h_{zkp} be a random oracle. If a class A1 attacker $(\overline{q}, \epsilon)$-breaks the* D-ELG-DEL$[\mathcal{E}, \mathcal{D}]$ *system, then there is an attacker that $(q_{\mathsf{dvc}}, \epsilon')$-breaks the underlying* ELGL *scheme with $\epsilon' \approx \epsilon$.*

Proof: Assume a class A1 attacker A $(\overline{q}, \epsilon)$-breaks the D-ELG-DEL system. Then we show how to construct an attacker A^* that $(q_{\mathsf{dvc}}, \epsilon')$-breaks the underlying ELGL scheme with $\epsilon' \approx \epsilon$. Suppose A^* is given public key $<g, p, q, y>$ for the ELGL scheme, and corresponding test and decryption oracles.

A^* runs the following simulation of the D-ELG-DEL system. A^* sets the dvc public key to $<g, p, q, y>$, and chooses $\pi_0 \leftarrow_R \mathcal{D}$. It generates all server key pairs $\{(pk_{\mathsf{svr}}, sk_{\mathsf{svr}})\}_{\mathsf{svr} \in U}$ and gives $\{pk_{\mathsf{svr}}\}_{\mathsf{svr} \in U}$ to A. A^* then constructs the authorization record $<\mathsf{svr}_0, pk_{\mathsf{svr}_0}, \tau, t, v, x_1, a>$

as normal, except setting $x_2 \leftarrow_R \mathbb{Z}_q$. A^* stores this authorization record and this corresponding value of x_2; x_2 is called the "correct server share" for τ. A^* then responds to oracle queries from A as follows:

- A^* responds to svr and disable oracle queries as in the real system.
- 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 $\hat{c}$ is the response.
- A^* responds to h_{zkp} queries as a normal random oracle.
- A^* responds to a $\mathsf{startDecr}(c, \mathsf{svr})$ query as in the real system. A^* responds to a $\mathsf{finishDecr}(\eta, e, s)$ query corresponding to a $\mathsf{startDecr}(c, \mathsf{svr})$ query, with $w \leftarrow \mathsf{select}(c)$, by looking up the authorization record $<\mathsf{svr}, pk_{\mathsf{svr}}, \tau, t, v, x_1, a>$ and the correct server share x_2 for τ, computing $\nu \leftarrow \eta \oplus \rho$ (for the ρ value from the $\mathsf{startDecr}(c, \mathsf{svr})$ query) and checking that $\nu^q \equiv_p 1$ and $e = h_{\mathsf{zkp}}(<\nu, w^s\nu^{-e} \bmod p, g^{s-ex_2} \bmod p>)$. If this is false, A^* simulates a dvc abort. Otherwise, A^* queries the ELGL decryption oracle with c (note that by definition of success for a class A1 attacker, this will be distinct from $\hat{c}$) 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 $\mathsf{finishDecr}(\eta, e, s)$ query.
- A^* responds to revoke queries as in the real system. A^* responds to a $\mathsf{startDel}(\mathsf{svr}, \mathsf{svr}')$ as in the real system. A^* responds to a $\mathsf{finishDel}(\delta', \eta)$ query corresponding to a $\mathsf{startDel}(\mathsf{svr}, \mathsf{svr}')$ query by looking up the authorization record $<\mathsf{svr}, pk_{\mathsf{svr}}, \tau, t, v, x_1, a>$ and the correct server share x_2 for τ, behaving as in the real protocol to generate a new authorization record $<\mathsf{svr}', pk_{\mathsf{svr}'}, \tau', t, v', x_1', a'>$, and storing $x_2' \leftarrow x_2 - x_{21}' + x_{12}' \bmod q$ as the correct server share for τ'.

Finally, if A outputs a bit b for the D-ELG-DEL system, A^* outputs the same bit b for the underlying ELGL scheme.

Now, to analyze the simulation, we first show that if $e = h_{\mathsf{zkp}}(<\nu, w^s\nu^{-e} \bmod p, g^{s-ex_2} \bmod p>)$ and $\nu^q \equiv_p 1$ in a $\mathsf{finishDecr}(\eta, e, s)$ query, then with overwhelming probability $\nu \equiv_p w^{x_2}$. By the forking lemma [18], with non-negligible probability we can obtain another triple $<\nu, \hat{e}, \hat{s}>$ satisfying $\hat{e} = h_{\mathsf{zkp}}(<\nu, w^{\hat{s}}\nu^{-\hat{e}} \bmod p, g^{\hat{s}-\hat{e}x_2} \bmod p>)$ where $\hat{e} \not\equiv_q e$ and

$$\begin{aligned} w^s\nu^{-e} &\equiv_p w^{\hat{s}}\nu^{-\hat{e}} \\ g^{s-ex_2} &\equiv_p g^{\hat{s}-\hat{e}x_2} \end{aligned}$$

Since $\nu^q \equiv_p 1$, $\nu \equiv_p w^{\hat{x}_2}$ for some $\hat{x}_2$, and thus these imply

$$\begin{aligned} s - \hat{x}_2 e &\equiv_q \hat{s} - \hat{x}_2\hat{e} \\ s - x_2 e &\equiv_q \hat{s} - x_2\hat{e} \end{aligned}$$

and thus that $(\hat{x}_2 - x_2)e \equiv_q (\hat{x}_2 - x_2)\hat{e}$. Since $e \not\equiv_q \hat{e}$, we must have $\hat{x}_2 \equiv_q x_2$. It follows that $\nu \equiv_p w^{x_2}$ with overwhelming probability. In this case the response by the simulation for the device $\mathsf{finishDecr}(\eta, e, s)$ would be the same as the response in the real protocol, since both would return the correct plaintext.

Therefore, the above simulation is statistically indistinguishable from D-ELG-DEL from A's perspective. So, if A succeeds with probability $\frac{1}{2} + \epsilon$ in D-ELG-DEL, then it succeeds with probability $\frac{1}{2} + \epsilon' \approx \frac{1}{2} + \epsilon$ in this simulation, and thus A^* succeeds with probability $\frac{1}{2} + \epsilon'$ in the underlying ELGL scheme. ▯

Theorem 6.2 *Let h, f, and h_{zkp} be random oracles. Also suppose that f_v (for random v) and h have negligible probabilities of collision over $\mathcal{D}$. If a class A3 attacker $(\overline{q}, \epsilon)$-breaks the* D-ELG-DEL$[\mathcal{E}, \mathcal{D}]$ *system with $\epsilon = \frac{q_h + q_f}{|\mathcal{D}|} + \psi$, then there is an attacker that $(q_{\mathsf{dvc}}, \epsilon')$-breaks the underlying* ELGL *scheme with $\epsilon' \approx \psi$.*

Proof: Assume a class A3 attacker A $(\overline{q}, \epsilon)$-breaks the D-ELG-DEL$[\mathcal{E}, \mathcal{D}]$ system with $\epsilon = \frac{q_h + q_f}{|\mathcal{D}|} + \psi$. Then we show how to construct an attacker A^* that $(q_{\mathsf{dvc}}, \epsilon')$-breaks the underlying ELGL scheme with $\epsilon' \approx \psi$. Suppose A^* is given public key $<g, p, q, y>$ for the ELGL scheme, and corresponding test and decryption oracles. We construct a simulation of the D-ELG-DEL system as in the proof of Theorem 5.1, except that the simulation aborts if the h oracle or f oracle is queried with π_0. This simulation is statistically indistinguishable from the real system (from F's viewpoint) unless the simulation aborts, the probability of which is exactly that of an offline dictionary attack (i.e., bounded by at most $\frac{q_h + q_f}{|\mathcal{D}|}$ plus the probability of a collision of one of the random oracles over $\mathcal{D}$). So since A succeeds with probability $\frac{1}{2} + \epsilon$ in the real system with $\epsilon = \frac{q_h + q_f}{|\mathcal{D}|} + \psi$, A succeeds with probability at least $\frac{1}{2} + \epsilon'$ in the simulation, where $\epsilon' \approx \psi$. ▯

Theorem 6.3 *Suppose that h_{zkp} is a random oracle, that $\{f_v\}_v$ is a pseudorandom function family, and that f_v (for random v) has a negligible probability of collision over $\mathcal{D}$. If a class A2 attacker $(\overline{q}, \epsilon)$-breaks the* D-ELG-DEL$[\mathcal{E}, \mathcal{D}]$ *system where $\epsilon = \frac{q_{\mathsf{svr}}}{|\mathcal{D}|} + \psi$, then either there is an attacker A^* that $(3q_{\mathsf{svr}}, \epsilon')$-breaks $\mathcal{E}$ where $\epsilon' \approx \frac{\psi}{2|U|(2q_{\mathsf{dvc}}+1)}$, or there is an attacker A^{**} that $(q_{\mathsf{dvc}}, \epsilon'')$-breaks the underlying* ELGL *scheme with $\epsilon'' \approx \frac{\psi}{2}$.*

Proof: Assume a class A2 attacker A $(\overline{q}, \epsilon)$-breaks the D-ELG-DEL$[\mathcal{E}, \mathcal{D}]$ system with $\epsilon = \frac{q_{\mathsf{svr}}}{|\mathcal{D}|} + \psi$. Define a server to be *good* if A never compromises it. (Recall that we consider static adversaries only, and so the good servers are determined before the system begins.) Let Real' be the D-ELG-DEL system in which all instances of f_v used for servers that are not good are replaced by perfectly random functions, and say A succeeds in the Real' system with probability $\frac{1}{2} + \frac{q_{\mathsf{svr}}}{|\mathcal{D}|} + \psi'$. Then by the pseudorandomness of f_v, $\psi' \approx \psi$. Now we construct a simulator Sim for Real' that takes a public key $<g, p, q, y>$ for the ELGL scheme, and corresponding test and decryption oracles as input. If attacker A *wins* (as defined below) against Sim with probability greater than $\frac{1}{2} + \frac{q_{\mathsf{svr}}}{|\mathcal{D}|} + \frac{\psi}{2}$, then we will be able to construct an attacker that $(q_{\mathsf{dvc}}, \epsilon'')$-breaks the underlying ElGL scheme with $\epsilon'' \approx \frac{\psi}{2}$. If, on the other hand, A wins against Sim with probability at most $\frac{1}{2} + \frac{q_{\mathsf{svr}}}{|\mathcal{D}|} + \frac{\psi}{2}$, then we will construct an attacker that $(3q_{\mathsf{svr}}, \epsilon')$-breaks $\mathcal{E}$ where $\epsilon' \approx \frac{\psi}{2|U|(2q_{\mathsf{dvc}}+1)}$. We say that A *wins* against Sim if A successfully guesses the bit b used in the test oracle query or if A makes a *successful online password guess*. The latter happens if A makes a query to a good svr with input (γ, δ, τ), such that if $<a, *, *, *, *, *, *> \leftarrow D_{sk_{\mathsf{svr}}}(\tau)$ then $\delta = \mathsf{mac}_a(<\gamma, \tau>)$, and either

- τ is a ticket stored on the device for svr, v is stored in the authorization record with τ, and either
  - for a $\mathsf{handleDecr}$ query to svr, γ is not generated by a device $\mathsf{startDecr}(*, \mathsf{svr})$ query, $<*, \beta, *> \leftarrow D_{sk_{\mathsf{svr}}}(\gamma)$, and $\beta = f(v, \pi_0)$,
  - for a $\mathsf{handleDel}$ query to svr, γ is not generated by a device $\mathsf{startDel}(\mathsf{svr}, *)$ query, $<\beta, *, *, *> \leftarrow D_{sk_{\mathsf{svr}}}(\gamma)$, and $\beta = f(v, \pi_0)$; or
- τ is not a ticket stored on the device for svr, γ was generated by a device $\mathsf{startDecr}(*, \mathsf{svr})$ or $\mathsf{startDel}(\mathsf{svr}, *)$ query using a record $<\mathsf{svr}, *, *, *, v, *, *>$, $<*, b', *, *, *, *, *> \leftarrow D_{sk_{\mathsf{svr}}}(\tau)$, and $b' = f(v, \pi_0)$.

We now define Sim. Below, we say that Sim *zeroes* a ticket τ for svr if Sim generates $\tau \leftarrow E_{pk_{\mathsf{svr}}}(0^{4\kappa+2\lambda+\ell(\lambda)})$, and we call the values $<a, b, g, p, q, x_{12}, \zeta>$ present when Sim creates τ the *zeroed inputs* to τ. Similarly, Sim zeroes a value γ for svr if Sim generates $\gamma \leftarrow E_{pk_{\mathsf{svr}}}(0^{|c|+\kappa+\lambda})$ (respectively, $\gamma \leftarrow E_{pk_{\mathsf{svr}}}(0^{3\kappa+\lambda+\ell(\lambda)})$) in a $\mathsf{startSign}(m, \mathsf{svr})$ (resp., $\mathsf{startDel}(\mathsf{svr}, \mathsf{svr}')$) oracle query, and the values $<c, \beta, \rho>$ (resp., $<\beta, pk_{\mathsf{svr}'}, \rho, \alpha>$) are its *zeroed inputs*.

Sim is given an ElGL public key $<g, p, q, y>$, and corresponding test and decryption oracles. Sim gives $<g, p, q, y>$ to A as the device's ElGL public key. Then Sim generates all servers' key pairs $\{(pk_{\mathsf{svr}}, sk_{\mathsf{svr}})\}_{\mathsf{svr} \in U}$, and gives $\{pk_{\mathsf{svr}}\}_{\mathsf{svr} \in U}$ to F. Next Sim generates $\pi_0 \leftarrow_R \mathcal{D}$ and the data $<a, b, g, p, q, 0, \zeta>$ for the ticket τ in the normal way, except that x_2 is chosen as $x_2 \leftarrow_R \mathbb{Z}_q$. If svr_0 is good, then Sim zeroes τ for svr_0, and else Sim sets $\tau \leftarrow E_{pk_{\mathsf{svr}_0}}(<a, b, g, p, q, 0, \zeta>)$. x_2 is called the "correct server share" for τ.

Sim responds to oracle queries as follows (using a truly random function for f, as in Real').

- Sim responds to queries to the test oracle as in Theorem 6.1.
- Sim responds h_{zkp} queries as a normal random oracle.
- Sim responds to a $\mathsf{dvc.comp}$ query by giving all authorization records stored on the device to A. Sim responds to a $\mathsf{svr.comp}$ query by giving sk_{svr} to A.
- Sim responds to a svr $\mathsf{disable}(t')$ query by storing $u' = h_{\mathsf{dsbl}}(t')$. Subsequently, any ticket τ of the following form is considered disabled at svr: either τ is zeroed for svr with zeroed inputs $<*, *, *, \zeta, *>$ or $<*, *, *, \zeta, *> \leftarrow D_{sk_{\mathsf{svr}}}(\tau)$; and $<u', *> \leftarrow D_{sk_{\mathsf{svr}}}(\zeta)$.

- Sim responds to a svr $\mathsf{handleDecr}(\gamma, \delta, \tau)$ or $\mathsf{handleDel}(\gamma, \delta, \tau)$ query for a τ that has not been disabled at svr as a normal server would, except for the following changes:
  - If τ or γ was zeroed for svr, then its zeroed inputs are used in the $\mathsf{handleDecr}$ or $\mathsf{handleDel}$ processing. Otherwise, their actual decryptions using sk_{svr} are used.
  - Sim aborts in the event of a successful password guess.
- Sim responds to a dvc $\mathsf{startDecr}(c, \mathsf{svr})$ query as in Theorem 6.1, except if svr is good, it zeroes γ. Sim responds to a dvc $\mathsf{finishDecr}(\eta, e, s)$ query as in Theorem 6.1.
- Sim responds to a dvc $\mathsf{startDel}(\mathsf{svr}, \mathsf{svr}')$ query as in Theorem 6.1, except if svr is good, it zeroes γ. Sim responds to a dvc $\mathsf{finishDel}(\delta', \eta)$ query corresponding to a $\mathsf{startDel}(\mathsf{svr}, \mathsf{svr}')$ query as in Theorem 6.1, except if svr' is good it zeroes τ'.

Suppose that the probability of A winning against Sim is more than $\frac{1}{2} + \frac{q_{\mathsf{svr}}}{|\mathcal{D}|} + \frac{\psi}{2}$. Since all instances of f_v for non-good servers are replaced by random functions, and all τ or γ ciphertexts encrypted under the public keys of good servers are zeroed, A obtains no information on the password from Sim, and thus the probability of A making a successful online password guess is at most $\frac{q_{\mathsf{svr}}}{|\mathcal{D}|}$ plus the probability of a collision in f_v over $\mathcal{D}$ for one of q_{dvc} random v's. So, A succeeds in guessing the bit b chosen by the test oracle with probability $\frac{1}{2} + \epsilon''$ where $\epsilon'' \approx \frac{\psi}{2}$. An attacker A^{**} for the the underlying ElGL scheme can thus run Sim for A and output the bit b selected by A to $(q_{\mathsf{dvc}}, \epsilon'')$-break the underlying ElGL scheme.

Now assume that the probability of A winning against Sim is at most $\frac{1}{2} + \frac{q_{\mathsf{svr}}}{|\mathcal{D}|} + \frac{\psi}{2}$. Since A succeeds in Real' with probability $\frac{1}{2} + \frac{q_{\mathsf{svr}}}{|\mathcal{D}|} + \psi'$, it wins in the Real' with at least that probability. Then we construct an attacker A^* that breaks $\mathcal{E}$ with probability $\epsilon' \approx \frac{\psi}{2|U|(2q_{\mathsf{dvc}}+1)}$. Our attacker A^* is given a public key pk from $\mathcal{E}$ and corresponding decryption oracle, and runs a simulation of the Real' system for A, using a device signing key $<g, p, q, y>$ and private key $<g, p, q, x>$ that it generates itself.

First consider a simulator that gives pk to A as the public key pk_{svr} of some svr that is good, and then simulates Real' exactly, except for aborting on a successful password guess and using a decryption oracle to decrypt messages encrypted under key pk by the adversary. There will be at most $3q_{\mathsf{svr}}$ of these. (Note that the decryptions of any τ, γ or ζ generated by the simulator would already be known to the simulator.) This simulation would be perfectly indistinguishable from Real' to A (at least until A wins). Now consider the same simulation, but with all τ and γ values for good servers generated by the device zeroed. (Naturally, the server pretends the encryptions are of the normal messages, not strings of zeros.) The latter simulation is statistically indistinguishable from Sim. Thus, the probability of A winning in the latter simulation is at most $\frac{1}{2} + \frac{q_{\mathsf{svr}}}{|\mathcal{D}|} + \frac{\psi}{2}$ plus a negligible term due to the fact that the latter simulation is not perfectly indistinguishable from Sim, 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 $j \in \{0, \ldots, 2q_{\mathsf{dvc}} + 1\}$ correspond to the first j τ-ciphertexts or γ-ciphertexts (generated by A^*) to good servers being of the normal messages (and all ζ ciphertexts being of the normal messages), and the remainder being encryptions of strings of 0's, and let p_j be the probability of A winning in experiment j. Then the average value for $i \in \{0, \ldots, 2q_{\mathsf{dvc}}\}$ of $p_{i+1} - p_i$ is at least $\approx \frac{\psi}{2(2q_{\mathsf{dvc}}+1)}$. Therefore, to construct A^*, we simply have A^* choose a random value $i \in \{0, \ldots, 2q_{\mathsf{dvc}} + 1\}$, assign $pk_{\mathsf{svr}} \leftarrow pk$ for a random good server svr, and run experiment i as above, but if the $(i+1)^{\text{st}}$ encryption to be generated by the simulator is to use pk_{svr}, it calls the test oracle for this encryption, 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 it believes X_0 was encrypted by the test oracle), and 1 otherwise. By the analysis above, A^* breaks $\mathcal{E}$ with probability $\epsilon' \approx \frac{\psi}{2|U|(2q_{\mathsf{dvc}}+1)}$. □

Theorem 6.4 *Suppose that h_{zkp} is a random oracle. If a class A4 attacker $(\overline{q}, \epsilon)$-breaks the* D-ElG-Del$[\mathcal{E}, \mathcal{D}]$ *system, then there is either an attacker A^* that $(3q_{\mathsf{svr}}, \frac{\epsilon}{2|U|(q_{\mathsf{svr}}+1)})$-breaks $\mathcal{E}$, or an attacker A^{**} that $(q_{\mathsf{svr}}, \frac{\epsilon}{2})$-breaks the underlying* ElGL *scheme.*

Proof: Assume a class A4 attacker A $(\overline{q}, \epsilon)$-breaks the D-ElG-Del$[\mathcal{E}, \mathcal{D}]$ system. Consider the following simulation Sim of D-ElG-Del for A. Sim is given an ElGL public key $<g, p, q, y>$ and corresponding test and decryption oracles as input. Sim sets the device public key to $<g, p, q, y>$ and chooses $\pi_0 \leftarrow_R \mathcal{D}$. It generates all server key pairs $\{(pk_{\mathsf{svr}}, sk_{\mathsf{svr}})\}_{\mathsf{svr} \in U}$, and gives $\{pk_{\mathsf{svr}}\}_{\mathsf{svr} \in U}$ to A. Sim then constructs τ as in the real system, except with $x_2 \leftarrow 0$ and $\zeta \leftarrow E_{pk_{\mathsf{svr}_0}}(0^{2\kappa})$. Sim then saves the authorization record $<\mathsf{svr}_0, pk_{\mathsf{svr}_0}, \tau, t, v, x_1, a>$. For the ciphertext ζ, the tuple $<pk_{\mathsf{svr}_0}, \zeta, x_1>$ is recorded on an *offset list*. Sim then responds to oracle queries from A as follows:

- Sim responds to queries to the test oracle as in Theorem 6.1.
- Sim responds to dvc oracle queries as in the real system.
- Sim responds to a svr $\mathsf{handleDel}(\gamma, \delta, \tau)$ query as svr would normally, except for the following changes. Sim runs normally until either the decryption of ζ or until svr aborts. If svr does not abort, then Sim has computed $<a, b, g, p, q, x_{12}, \zeta> \leftarrow D_{sk_{\mathsf{svr}}}(\tau)$ and $<\beta, pk_{\mathsf{svr}'}, \rho, \alpha> \leftarrow D_{sk_{\mathsf{svr}}}(\gamma)$. Now Sim examines the offset list with pk_{svr} and ζ. If for some x_1^+, $<pk_{\mathsf{svr}}, \zeta, x_1^+>$ appears in the offset list, then Sim causes the server to abort if $u = h_{\mathsf{dsbl}}(t)$ is marked as disabled at svr, and otherwise

  – computes $\zeta' \leftarrow E_{pk_{\mathsf{svr}}}(0^{2\kappa})$,

  – computes $x_1 \leftarrow x_1^+ - x_{12} \bmod q$,

  – computes $x'_{21} \leftarrow \mathbb{Z}_q$ (i.e., as normal), and

  – stores $<pk_{\mathsf{svr}'}, \zeta', x_1 + x'_{21}>$ on the offset list.

  Sim then completes the computation of η and δ' and responds. If $<pk_{\mathsf{svr}}, \zeta, *>$ does not appear on the offset list, then Sim responds as svr would normally.

- Sim responds to a svr $\mathsf{handleDecr}(\gamma, \delta, \tau)$ query as svr would normally, except for the following changes: Sim runs normally until either the decryption of ζ or until svr aborts. If svr does not abort, then Sim has computed $<a, b, g, p, q, x_{12}, \zeta> \leftarrow D_{sk_{\mathsf{svr}}}(\tau)$ and $<c, \beta, \rho> \leftarrow D_{sk_{\mathsf{svr}}}(\gamma)$. Now Sim examines the offset list with pk_{svr} and ζ. If for some x_1^+, $<pk_{\mathsf{svr}}, \zeta, x_1^+>$ appears in the offset list, then Sim causes svr to abort if u is marked as disabled at svr, and otherwise

  – computes $x_1 \leftarrow x_1^+ - x_{12}$,

  – queries the decryption oracle on c to obtain z,

  – computes $\nu \leftarrow zw^{-(x_0+x_1)} \bmod p$,

  – generates $e \leftarrow_R \mathbb{Z}_q$ and $s \leftarrow_R \mathbb{Z}_q$, and

  – backpatches h_{zkp}:

  $$h_{\mathsf{zkp}}(<\nu, (c_2)^s \nu^{-e} \bmod p, g^s (y g^{-(x_0+x_1)})^{-e} \bmod p>) \quad \leftarrow \quad e$$

  Sim then completes the computation of η and responds. If $<pk_{\mathsf{svr}}, \zeta, *>$ does not appear on the offset list, then Sim responds as svr would normally.

  We note that once u is marked as disabled at all servers, there will never be a need to call the decryption oracle again.

- Sim responds to a svr $\mathsf{disable}(t')$ query by marking $u' = h_{\mathsf{dsbl}}(t')$ as disabled at svr.

- Sim responds to a dvc comp query by returning the authorization records on dvc. Note that there are no svr comp queries, because the attacker is in class A4.

If A succeeds with probability greater than $\frac{1}{2} + \frac{\epsilon}{2}$ in Sim, then there is an attacker A^{**} for the underlying ELGL scheme that succeeds with probability $\frac{1}{2} + \frac{\epsilon}{2}$. A^{**} is given the ELGL public key $<g, p, q, y>$ and corresponding test and decryption oracles. A^{**} runs Sim for A and outputs the bit b chosen by A.

Now suppose that A succeeds in Sim with probability at most $\frac{1}{2}+\frac{\epsilon}{2}$. We use a hybrid argument to construct an attacker A^* that breaks $\mathcal{E}$ with probability at least $\frac{\epsilon}{2|U|(q_{\mathsf{svr}}+1)}$, as follows. A is given a public key pk from $\mathcal{E}$ with corresponding decryption and test oracles, and sets $pk_{\mathsf{svr}^*} \leftarrow pk$ for a randomly chosen svr^*. In addition, A^* generates $(<g, p, q, y>, <g, p, q, x>) \leftarrow G_{\mathsf{ElGL}}(1^\lambda)$ and sets the device's public key to $<g, p, q, y>$. A^* then chooses an index $i \leftarrow_R \{0, \ldots, q_{\mathsf{svr}}\}$. A^* computes x_0, x_1, x_2, t, u, v, a, and b normally, but sets ζ as follows: if $i = 0$ and $\mathsf{svr}_0 = \mathsf{svr}^*$, then it sets ζ to be the output of the test oracle for pk queried with $X_0 = <u, x_2>$ and $X_1 = 0^{2\kappa}$ and records the tuple $<pk_{\mathsf{svr}_0}, \zeta, x_2>$ on an *offset list*.[4] If $i > 0$ or $\mathsf{svr}_0 \neq \mathsf{svr}^*$, then it sets ζ normally and records the tuple $<pk_{\mathsf{svr}_0}, \zeta, x_2>$ on the offset list. After this, A^* proceeds normally to complete the authorization record $<\mathsf{svr}_0, pk_{\mathsf{svr}_0}, \tau, t, v, x_1, a>$. A^* then simulates D-ELG-DEL for A as follows.

[4]The third value in the tuple is the value that is supposed to be encrypted in ζ (but may not be the actual value that is encrypted).

- A^* responds to queries to the test oracle as in Theorem 6.1.

- A^* responds to dvc oracle queries as in the real system.

- A^* responds to a svr $\mathsf{handleDel}(\gamma, \delta, \tau)$ query as follows. If either γ or τ are the output of the test oracle (if the test oracle has been queried), then svr aborts. Otherwise, A^* runs normally—using the decryption oracle for pk to decrypt γ and τ if $\mathsf{svr} = \mathsf{svr}^*$—until either the decryption of ζ or until svr aborts. If svr does not abort, suppose this is the j-th $\mathsf{handleDel}$ query (to any server) in which the server did not abort. At this point A has computed $<a, b, g, p, q, x_{12}, \zeta> \leftarrow D_{sk_{\mathsf{svr}}}(\tau)$ and $<\beta, pk_{\mathsf{svr}'}, \rho, \alpha> \leftarrow D_{sk_{\mathsf{svr}}}(\gamma)$ (or computed them using the decryption oracle). Now A examines the offset list with pk_{svr} and ζ:

  - If $<pk_{\mathsf{svr}}, \zeta, *>$ does not appear in the offset list, then if $pk \neq pk_{\mathsf{svr}}$, A^* decrypts ζ to obtain $<u', x_{22}>$, and otherwise A^* calls the decryption oracle on ζ to obtain $<u', x_{22}>$. A^* causes svr to abort if u' is marked as disabled at svr, and otherwise A continues as svr would normally.

  - If for some x_{22}, $<pk_{\mathsf{svr}}, \zeta, x_{22}>$ appears in the offset list, then A^* has svr abort if $u = h_{\mathsf{dsbl}}(t)$ is marked as disabled at svr, and otherwise proceeds as follows: A^* sets $x_2 \leftarrow x_{12} + x_{22}$, computes x'_{21} and x'_{22} as normal, and alters the computation of ζ' as follows. If $j < i$, then A^* computes $\zeta' \leftarrow E_{pk_{\mathsf{svr}'}}(<u, x'_{22}>)$ as normal and records $<pk_{\mathsf{svr}'}, \zeta', x'_{22}>$ in the offset list. If $j = i$ and $\mathsf{svr}' = \mathsf{svr}^*$, then A calls the test oracle for pk with $X_0 = <u, x'_{22}>$ and $X_1 = 0^{\lambda+2\kappa}$, obtains the response ζ', and records $<pk_{\mathsf{svr}'}, \zeta', x'_{22}>$ in the offset list. If $j > i$, or $j = i$ and $\mathsf{svr}' \neq \mathsf{svr}^*$, then A sets $\zeta' \leftarrow E_{pk_{\mathsf{svr}}}(0^{2\kappa})$ and records $<pk_{\mathsf{svr}'}, \zeta', x'_{22}>$ in the offset list. In all cases, A^* then computes η and δ' normally and responds.

- A^* responds to a svr $\mathsf{handleDecr}(\gamma, \delta, \tau)$ query as follows. If either γ or τ are the output of the test oracle (if the test oracle has been queried), then svr aborts. Otherwise, A^* runs normally—using the decryption oracle for pk to decrypt γ and τ if $\mathsf{svr} = \mathsf{svr}^*$—until either the decryption of ζ or until svr aborts. If svr does not abort, then Sim has computed $<a, b, g, p, q, x_{12}, \zeta> \leftarrow D_{sk_{\mathsf{svr}}}(\tau)$ and $<c, \beta, \rho> \leftarrow D_{sk_{\mathsf{svr}}}(\gamma)$ (or computed them using the decryption oracle). Now A^* examines the offset list with pk_{svr} and ζ:

  - If $<pk_{\mathsf{svr}}, \zeta, *>$ does not appear in the offset list, then if $pk \neq pk_{\mathsf{svr}}$, A^* decrypts ζ to obtain $<u', x_{22}>$, and otherwise A^* calls the decryption oracle on ζ to obtain $<u', x_{22}>$. If u' is marked as disabled at svr, then A^* simulates a svr abort, and otherwise A^* continues as svr would normally.

  - If for some x_{22}, $<pk_{\mathsf{svr}}, \zeta, x_{22}>$ appears in the offset list, then A^* causes svr to abort if $u = h_{\mathsf{dsbl}}(t)$ is marked as disabled at svr. If u is not marked as disabled, then A^* sets $x_2 \leftarrow x_{12} + x_{22}$, and responds as svr would normally.

- A^* responds to a svr $\mathsf{disable}(t')$ query by marking $u' = h_{\mathsf{dsbl}}(t')$ as disabled at svr.

- A^* responds to a $\mathsf{dvc\ comp}$ query by returning the authorization records on dvc. Note that there are no $\mathsf{svr\ comp}$ queries, because the attacker is in class A4.

Finally, A^* outputs the bit b output by A.

Let experiment $j \in \{0, \ldots, q_{\mathsf{svr}} + 1\}$ correspond to the first j ζ-ciphertexts (generated by A^*), i.e., the ciphertexts internal to the tickets, being of normal messages, and the remainder being encryptions of zero. Let p_j be the probability of A succeeding in experiment j. Note that experiment 0 is perfectly indistinguishable from Sim, and experiment $q_{\mathsf{svr}} + 1$ is perfectly indistinguishable from the real system. Therefore $p_0 \leq \frac{1}{2} + \frac{\epsilon}{2}$ and $p_{q_{\mathsf{svr}}+1} \geq \frac{1}{2} + \epsilon$. Note that A^* will run either experiment i or $i+1$, depending on the output of the test oracle, and if svr^* is the server that receives encryption request $i+1$. Then the average value for $i \in \{0, \ldots, q_{\mathsf{svr}}\}$ of $p_{i+1} - p_i$ is at least $\approx \frac{\epsilon}{2(q_{\mathsf{svr}}+1)}$, and the probability with which A^* breaks $\mathcal{E}$ is at least $\frac{\epsilon}{2|U|(q_{\mathsf{svr}}+1)}$. □

References

[1] M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations among notions of security for public-key encryption schemes. In *CRYPTO '98* (LNCS 1462), pp. 26–45, 1998.

[2] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In 1st *ACM Conf. on Comp. and Comm. Security*, pp. 62–73, 1993.

[3] M. Bellare and P. Rogaway. Optimal asymmetric encryption. In *EUROCRYPT '94* (LNCS 950), pp. 92–111, 1995.

[4] M. Bellare and P. Rogaway. The exact security of digital signatures—How to sign with RSA and Rabin. In *EUROCRYPT '96* (LNCS 1070), pp. 399–416, 1996.

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

[6] R. Cramer and V. Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *CRYPTO '98* (LNCS 1462), pp. 13–25, 1998.

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

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

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

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

[11] A. Herzberg, M. Jakobsson, S. Jarecki, H. Krawczyk, and M. Yung. Proactive public key and signature systems. In 4^{th} *ACM Conf. on Comp. and Comm. Security*, pp. 100-110, 1997.

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

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

[14] P. MacKenzie and M. K. Reiter. Networked cryptographic devices resilient to capture. DIMACS Technical Report 2001-19, May 2001. Extended abstract in *2001 IEEE Symp. on Security and Privacy*, May 2001.

[15] P. MacKenzie and M. K. Reiter. Delegation of cryptographic servers for capture-resilient devices (extended abstract). To appear in 8^{th} *ACM Conf. on Comp. and Comm. Security*, Nov. 2001.

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

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

[18] D. Pointcheval and J. Stern. Security proofs for signature schemes. In *EUROCRYPT '96* (LNCS 1070), pages 387–398, 1996.

[19] C. Rackoff and D. Simon. Noninteractive zero-knowledge proof of knowledge and chosen ciphertext attack. In *CRYPTO '91*, (LNCS 576), pp. 433–444, 1991.

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

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