Distrib. Comput. (2003) 16: 307–327
Digital Object Identifier (DOI) 10.1007/s00446-003-0098-4

DISTRIBUTED COMPUTING

Delegation of cryptographic servers for capture-resilient devices*

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

[1] Bell Labs, Lucent Technologies, Murray Hill, NJ, USA (e-mail: philmac@research.bell-labs.com)
[2] Carnegie Mellon University, Pittsburgh, PA, USA (e-mail: reiter@cmu.edu)

Received November 2001 / Accepted: January 2003

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., [18, 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., [9,15]).

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 [15], 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 [15] 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

* Extended abstract appears in *Proceedings of the* 8^{th} *ACM Symposium on Computer and Communications Security*, November 2001.

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 [15]. 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 [21] 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 [15] to accommodate delegation. One of the most fundamental modifications lies in how the password itself is protected: whereas the original systems of [15] 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 or even from gaining partial information about the password from the user's behavior (e.g., the ways in which the user mistypes it). Herein lies a fundamental tension: By the nature of the server's role, i.e., checking the user's password, the server will necessarily learn the *frequency* with which the user mistypes her password, which is a form of partial information about it. This tension is one significant source of differentiation from [15], both in new protocols that limit this information leakage to the frequency of mistypes, and in new proof techniques to enable the impact of this leakage to be isolated.

An extended abstract of this paper [17] 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 and enters it correctly) 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 on user input – and the user does not subsequently provide input to the device. 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. svr.disable: svr stops responding to any requests of the device (signing, decryption, or delegation).
5. dvc.comp: dvc is compromised (and captured).
6. svr.comp: svr is compromised.
7. π_0.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 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 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). Note that achieving G3 requires that π_0 play an additional role in the signing or decryption protocol than merely being checked by the server, since the adversary who compromised the server can choose to not perform such a check. It is not difficult to verify that these goals are a strict generalization of the goals of [15]; i.e., these goals reduce to those of [15] in the case $|U| = 1$.

2.2.1 Leakage of password information

A subtlety that pertains to A2 and A3 attackers is the need to prevent them from obtaining any useful information *before* dvc.comp occurs, with which they can later attack the system in ways not permitted by G2 or G3. In particular, either of these attackers may compromise servers and interact with 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.

before dvc.comp occurs. (If one of these already compromised servers has not been revoked when dvc.comp occurs, then the attacker is an A3 attacker.) During this period, the attacker can observe queries from dvc operated by the correct user, and if these queries leak any information about π_0, then the attacker can subsequently use this information to reduce the set of candidate passwords to a subset of $\mathcal{D}$. Therefore, the attacker will be able to violate G2 or G3 after dvc.comp. This is in contrast to an A1 attacker, for which G1 continues to hold even if the attacker performs π_0.comp.

Due to the role of a server in our protocol, some partial information about π_0, namely the frequency of password mistypes, necessarily leaks to an A2 or A3 attacker who has compromised a server used by dvc before dvc.comp occurs. In our proofs of security in Sects. 6 and 7, therefore, we factor out this limitation by requiring the attacker to specify when mistypes occur. Obviously, the attacker then gains no extra information from the simple fact that a mistype occurs, since the attacker decides when they occur.

2.2.2 Proactivity

Delegation offers 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.

2.3 Tools

Our systems for meeting the goals outlined in Sect. 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) we will need for the hash functions 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 $\{f(v)\}_v$ 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 make use of keyed hash functions of two forms in this paper, namely pseudorandom function families and message authentication codes (MACs). Below we denote a pseudorandom function family by $\{f(v)\}_v$ and a MAC family by $\{\mathsf{mac}(a)\}_a$. For some of our proofs we will require pseudorandom functions to behave as random oracles, though 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 $\ell(|m|)$. D takes a ciphertext c and a private 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 [20]. Practical examples can be found in [2,3,6].

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 private 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 1 if σ' is a valid signature for m for the corresponding private key, and otherwise returns 0. That is, $V(pk, m, \sigma') \in \{0,1\}$. 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 [15] 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 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 [15] 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 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 key pair of the device is $(pk_{\mathsf{dvc}}, sk_{\mathsf{dvc}}) \leftarrow G_{\mathrm{RSA}}(1^\lambda)$, where $pk_{\mathsf{dvc}} = <e, N>$, $sk_{\mathsf{dvc}} = <d, N, \phi(N)>$, $ed \equiv_{\phi(N)} 1$, N is the product of two $\lambda/2$-bit 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):

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

Note that it may not be necessary to return r if it can be determined from m and σ. Here, $r \leftarrow_R \{0,1\}^{\kappa_{sig}}$ denotes selecting an element of $\{0,1\}^{\kappa_{sig}}$ uniformly at random and assigning this selection to r. 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 $\sigma^e \equiv_N \mathsf{encode}(m, r)$.

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:

$$\begin{aligned} & a \leftarrow_R \{0,1\}^\kappa \\ & t \leftarrow_R \{0,1\}^\kappa \\ & u \leftarrow h_{\mathsf{dsbl}}(t) \\ & v_0 \leftarrow_R \{0,1\}^\kappa \\ & d_0 \leftarrow f_0(v_0, \pi_0) \\ & d_1 \leftarrow d - d_0 \bmod \phi(N) \\ & v_1 \leftarrow_R \{0,1\}^\kappa \\ & b \leftarrow f_1(v_1, \pi_0) \\ & \zeta \leftarrow E(pk_{\mathsf{svr}_0}, <u, d_1>) \\ & \tau \leftarrow E(pk_{\mathsf{svr}_0}, <a, b, 0, \zeta, N>) \end{aligned}$$

The values pk_{dvc} and u, and the *authorization record*

$$<\mathsf{svr}_0, pk_{\mathsf{svr}_0}, \tau, t, v_0, v_1, a> \tag{1}$$

are saved on stable storage in the device. All other values, including d, $\phi(N)$, π_0, b, d_0, d_1, and ζ, are deleted from the device. The τ value is the device's "ticket" that it uses to access svr_0. The u value is the "ticket identifier".

Several comments about this initialization are helpful in understanding the protocols in subsequent sections:

- d is broken into shares d_0 and d_1 such that $d_0 + d_1 \equiv_{\phi(N)} d$ [5] using a pseudorandom function family $\{f_0(v) : \mathcal{D} \rightarrow \{0,1\}^{\lambda+\kappa}\}_{v \in \{0,1\}^\kappa}$, where $\mathcal{D}$ is the space (the "dictionary") from which passwords are drawn. Since d_0 is not stored on dvc, an A3 adversary who captures dvc and svr_0 (before $\mathsf{dvc.revoke}(\mathsf{svr}_0)$ occurs) would need to succeed in an offline dictionary attack to find it. d_1 is encrypted for svr_0 and so remains hidden from an A2 or A4 attacker.
- As will be described in greater detail in Sect. 3.5, t can be sent to svr_0 to disable svr_0 for this device or, more precisely, for the ticket τ. svr_0 recognizes a ticket τ to which a disabling request bearing t pertains as any such that $<*, *, *, \zeta, *> \leftarrow D(sk_{\mathsf{svr}}, \tau)$, $<u, *> \leftarrow D(sk_{\mathsf{svr}}, \zeta)$, and $h_{\mathsf{dsbl}}(t) = u$. Since t is primarily needed after dvc is captured (by an A2 or A4 attacker), t should be copied off the device for use in disabling if the need arises.
- The ticket τ will be sent to svr within the context of the S-RSA-DEL signing and delegation protocols (see Sects. 3.3 and 3.4), and the server will inspect the contents of the ticket to extract its share d_1 of the device's private signing key. In anticipation of its own compromise, dvc might include a policy statement within τ and ζ 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 Sect. 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.
- The value a is a MAC key that will be used in the S-RSA-DEL signing and delegation protocols to enable svr_0 to distinguish between requests from dvc, where a is stored, and spurious requests from parties not possessing dvc. As will be described in more detail in Sect. 3.3, this is important for defending against denial-of-service attacks on the device. The value a is included in τ so that svr_0 can extract it upon decrypting τ.
- The value b is computed using the pseudorandom function family $\{f_1(v) : \mathcal{D} \rightarrow \{0,1\}^\kappa\}_{v \in \{0,1\}^\kappa}$, where $\mathcal{D}$ is the

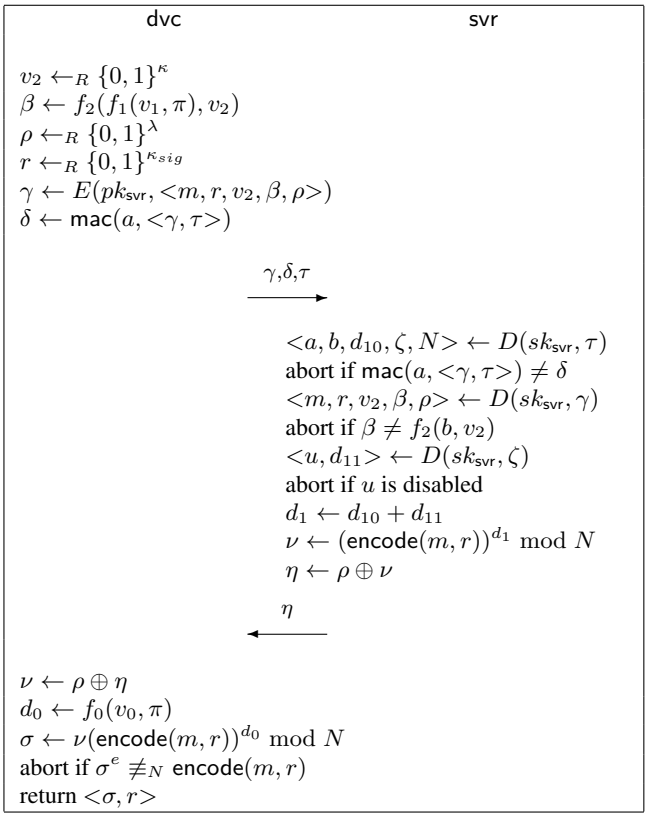

Fig. 1. S-RSA-DEL signature protocol

space from which passwords are drawn. b is included in τ to enable svr_0 to confirm that a request from dvc was originated by a user who knew π_0. In Sects. 3.3–3.4, b will be used as a key to another pseudorandom function $\{f_2(v) : \{0,1\}^\kappa \rightarrow \{0,1\}^\kappa\}_{v \in \{0,1\}^\kappa}$.

Each execution of the S-RSA-DEL delegation protocol of Sect. 3.4 to authorize a new server svr will yield a new authorization record stored on dvc. The values in this new authorization record have roles with respect to svr analagous to how the values in (1) are used in the signing and delegation protocols with svr_0.

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 $<\mathsf{svr}, pk_{\mathsf{svr}}, \tau, t, v_0, v_1, a>$, generated either in the initialization procedure of Sect. 3.2, or in the delegation protocol of Sect. 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 Fig. 1.

This protocol generates a signature for m by constructing $\mathsf{encode}(m,r)^{d_0+d_1} \bmod N$, where d_0 is derived from the user's password and d_1 is stored (partially encrypted) in τ. svr cooperates in this operation only after confirming that the v_2 and β values contained within the ciphertext γ are valid evidence that dvc holds the user's password, i.e., if $\beta = f_2(b, v_2)$ where b is extracted from τ. If svr can confirm this construction, then it generates $\nu = \mathsf{encode}(m,r)^{d_1} \bmod N$, and dvc multiplies ν by $\mathsf{encode}(m,r)^{d_0} \bmod N$ to get the desired result. It is important that dvc deletes π, d_0, ρ (used to encrypt ν), and the results of all intermediate computations when the protocol completes, and that it never stores 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 $<\mathsf{svr}, pk_{\mathsf{svr}}, \tau, t, v_0, v_1, a>$, a public key $pk_{\mathsf{svr}'}$ for another server $\mathsf{svr}' \in U$, and the input password π. (As described in Sect. 3.2, one could also input additional policy information here.) Recall that dvc also stores $pk_{\mathsf{dvc}} = <e, N>$. The protocol is (partially) described in Fig. 2, with one significant omission that is detailed below.

The overall goal of the protocol in Fig. 2 is to generate a new share d_1' for server svr', and new ticket τ' for the device to use with svr'. The new share d_1' for svr' is constructed as $d_1' = d_{10}' + d_{11}' = (d_0 - d_0' + \Delta) + (d_1 - \Delta)$, with the first and second terms being computed by dvc and svr, respectively. As a result, $d_0' + d_1' = d_0 + d_1$. Note that the two terms d_{10}' and d_{11}' are actually stored separately in the ticket, with d_{11}' being encrypted by the consenting server so the device can not determine the full share d_1' when it constructs the new ticket. Hiding d_1' from dvc is a central component of achieving G4, as otherwise an A4 attacker could reconstruct d after executing the delegation protocol just once. svr inserting u into ζ' is similarly important: if u were inserted into τ' by dvc, then an A4 attacker could change it and thereby prevent τ' from ever being disabled.

Note that in the protocol of Fig. 2, it is not possible for dvc to confirm the correctness of the response it receives from svr. This is in contrast to the protocol of Fig. 1, where dvc can check the resulting signature. There are two ways in which svr– or more specifically, an adversary controlling svr– could misbehave that are relevant here. The first is if svr fails to abort the protocol when $\beta \neq f_2(b, v_2)$, i.e., even if dvc is operating with an incorrect password $\pi \neq \pi_0$. In this case, the delegation protocol would complete, but it would create a useless authorization record $<\mathsf{svr}', pk_{\mathsf{svr}'}, \tau', t, v_0', v_1', a'>$ that could not be used with svr' to subsequently sign. But this is not the worst of it: with this as a possibility, we do not know how to prove property G2. As a result, to achieve a proof of G2, we stipulate that the protocol in Fig. 2 must (somehow)

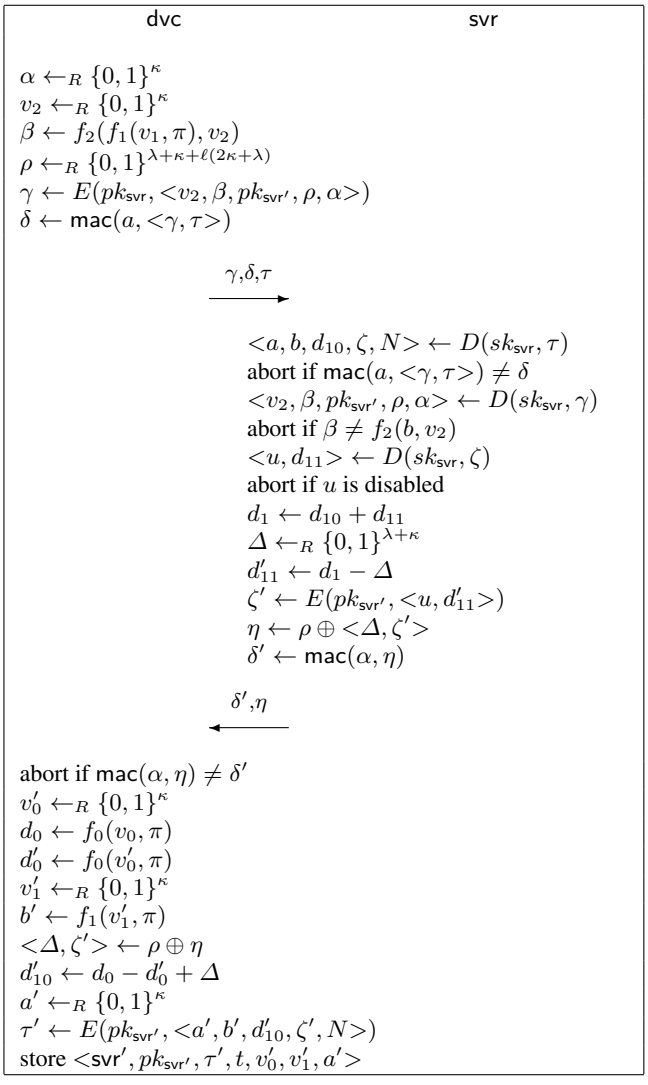

Fig. 2. S-RSA-DEL delegation protocol

be aborted by dvc before completing if $\pi \neq \pi_0$. Moreover, this must be achieved without adding any stored state to dvc, lest this additional state permit an offline dictionary attack to occur if dvc is captured. In practice, this can be achieved if dvc runs the protocol in Fig. 2 concurrently (or after) an instance of the signature protocol in Fig. 1, say on a random message m, with the same password π: If dvc obtains a correct signature on m, then it knows $\pi = \pi_0$ and can proceed safely with delegation.

The second way in which svr can misbehave is to simply produce an incorrect η. This is no obstacle to our proofs of G1–G4; it does, however, still yield a useless new authorization record, and we know of no defense to this denial-of-service attack. The next best thing to a defense, however, is to limit this vulnerability to svr only, i.e., to prevent a network attacker from fabricating η, and this is exactly the purpose of α in Fig. 2. α is a mac key used to authenticate svr's response to dvc.

Aside from these distinctions, this protocol borrows many components from the signature protocol of Fig. 1. For example, β, γ and δ all play similar roles in the protocol as they did in Fig. 1. And deletion is once again important: dvc must delete α, β, b', d'_{10}, d_0, d'_0, ρ, π, ζ', and all other intermediate computations at the completion of this protocol. Similarly, svr should delete α, β, b, d_{10}, d_{11}, Δ, d'_{11}, ρ, ζ', and all other intermediate results when it completes.

To relate this protocol to the system model of Sect. 2.1, and for our proofs in Sect. 6, we define the execution of the code before the first message in Fig. 2 to constitute a dvc.startDel(svr, svr′) event. Likewise, we define the execution of the code after the second message in Fig. 2 to constitute a dvc.finishDel(svr, svr′) event. The event dvc.revoke(svr), though not pictured in Fig. 2, can simply be defined as dvc deleting any authorization record $<\mathsf{svr}, pk_{\mathsf{svr}}, \tau, t, v_0, v_1, a>$ and halting any ongoing delegation protocols to authorize svr.

3.5 Key disabling

As in [15], 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 Sect. 2.1, 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, our prior work [15] 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.

In designing the D-ELG-DEL decryption and delegation protocols, we attempted to mimic the approach taken in S-RSA-DEL as closely as possible. However, substantive differences are required by the goal of decryption (versus signatures). Some of these differences relate primarily to the definition of security, which will be introduced in Sect. 4.1. Other differences impact the protocol more noticeably, however. For example, in S-RSA-DEL the device is protected against a cheating svr (an A1 attacker) partially because it can verify the proper behavior of svr, i.e., by verifying the resulting signature. Decryption is different, however, in that dvc cannot simply re-encrypt the result of the decryption protocol to verify that svr provided proper output; ElGamal encryption

is randomized (as is any secure encryption), and decryption does not reveal the random bits used in the encryption. Practically speaking, this problem can be overcome to some extent "outside" our decryption protocol, by requiring the ciphertext creator to embed redundancy into the plaintext that dvc can verify upon completion of the decryption protocol. This is less satisfying for several reasons, however: First, as a practical matter, a failed redundancy check does not distinguish svr misbehavior from misbehavior by the ciphertext creator. Second, we cannot prove the security of such a protocol. As a result, the approach we adopt here is one in which svr *proves* that it has performed its function correctly. Moreover, it proves this in zero-knowledge so as to leak no new information to dvc– which, in turn, is essential for proving security against an A4 attacker.

Our techniques, applied to decryption algorithms, only apply to ones that do not have a private validity check, that is, a validity check that uses the private key. (For example, OAEP [3] is a scheme that uses a private validity check.) We also note that while protocols for signature schemes based on discrete logarithms (e.g., DSA [14]) do not immediately follow from the protocol of this section, they can be achieved using more specialized cryptographic techniques, as corollaries of [16].

4.1 Preliminaries

For ElGamal encryption, the public key pair of the device is $(pk_{\mathsf{dvc}}, sk_{\mathsf{dvc}}) \leftarrow G_{\mathrm{ElG}}(1^\lambda)$ where $pk_{\mathsf{dvc}} = <g, p, q, y>$, $sk_{\mathsf{dvc}} = <g, p, q, x>$, 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 [15], we describe the D-ELG protocol using an abstract specification of "ElGamal-like" encryption. An *ElGamal-like encryption scheme* is an encryption scheme in which (i) the public and private keys are as above; and (ii) the decryption function D can be expressed in the following form:

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

Above, $\mathsf{valid}(c)$ tests the well-formedness of the ciphertext c; it returns 1 if well-formed and 0 otherwise. $\mathsf{select}(c)$ returns the argument w that is raised to the x-th power modulo p. $\mathsf{reveal}(z, c)$ generates the plaintext m using the result z of that computation. For example, in original ElGamal encryption, where $q = p-1$ and $c = <c_1, c_2> = <g^k \bmod p, my^k \bmod p>$ for some secret value $k \in \mathbb{Z}_q$, $\mathsf{valid}(<c_1, c_2>)$ returns 1 if $c_1, c_2 \in \mathbb{Z}_p^*$ and 0 otherwise; $\mathsf{select}(<c_1, c_2>)$ returns c_1; and $\mathsf{reveal}(z, <c_1, c_2>)$ returns $c_2 z^{-1} \bmod p$. We note, however, that the private key is *not* an argument to valid, select, or reveal; rather, the private key is used only in computing z. Using this framework, the D-ELG-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 A4 adversary who is presented with a ciphertext c *after* $\max_{\mathsf{svr} \in U}\{T(\mathsf{svr.disable})\}$ 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 [20], 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 [19]). There are, however, ElGamal-like encryption schemes that suffice to achieve even the latter, stronger security property, such as the proposals in [22]. When our protocol is instantiated with one of these (as in [15]), 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:

$a \leftarrow_R \{0,1\}^\kappa$
$t \leftarrow_R \{0,1\}^\kappa$
$u \leftarrow h_{\mathsf{dsbl}}(t)$
$v_0 \leftarrow_R \{0,1\}^\kappa$
$x_0 \leftarrow f_0(v_0, \pi_0)$
$x_1 \leftarrow x - x_0 \bmod q$
$v_1 \leftarrow_R \{0,1\}^\kappa$
$b \leftarrow f_1(v_1, \pi_0)$
$\zeta \leftarrow E(pk_{\mathsf{svr}_0}, <u, x_1>)$
$\tau \leftarrow E(pk_{\mathsf{svr}_0}, <a, b, g, p, q, 0, \zeta>)$

Here, we assume the pseudorandom function families f_0 and f_1 have the form $\{f_0(v) : \mathcal{D} \to \mathbb{Z}_q\}_{v \in \{0,1\}^\kappa}$ and $\{f_1(v) : \mathcal{D} \to \{0,1\}^\kappa\}_{v \in \{0,1\}^\kappa}$, where $\mathcal{D}$ is the space from which passwords are drawn. And, as in Sect. 3, the value b will be used as the key for another pseudorandom function family $\{f_2(v) : \{0,1\}^\kappa \to \{0,1\}^\kappa\}_{v \in \{0,1\}^\kappa}$.

The value pk_{dvc} and the authorization record $<\mathsf{svr}_0, pk_{\mathsf{svr}_0}, \tau, t, v_0, v_1, a>$, are saved on stable storage in the device. All other values, including b, u, x, x_0, x_1, 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 $<\mathsf{svr}, pk_{\mathsf{svr}}, \tau, t, v_0, v_1, a>$. Recall that dvc also stores $pk_{\mathsf{dvc}} = <g, p, q, y>$.

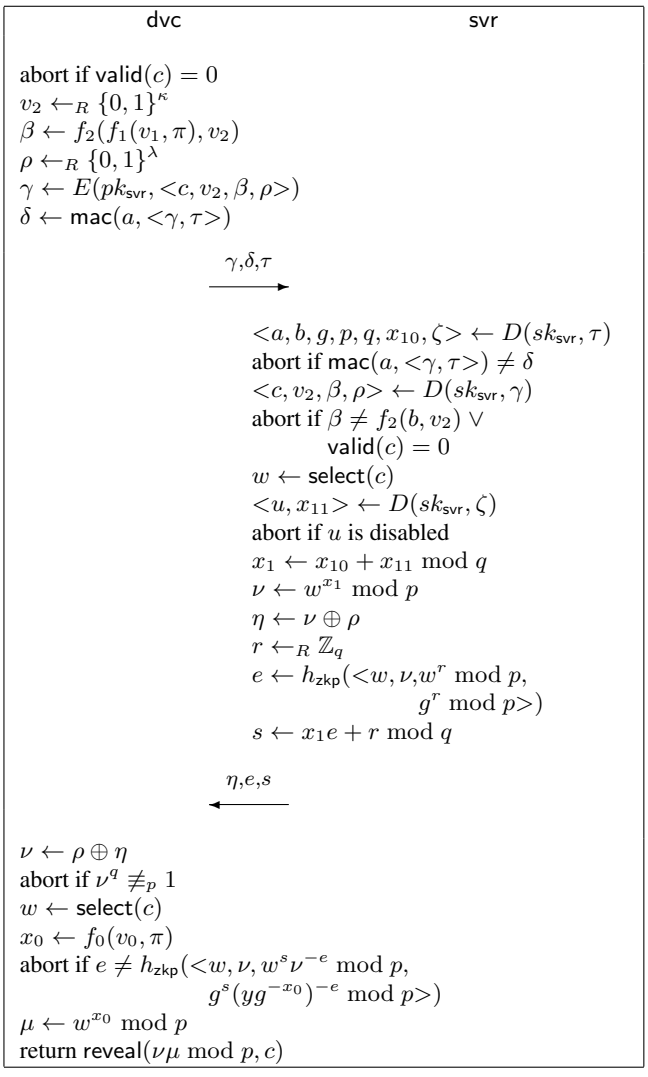

Fig. 3. D-ELG-DEL decryption protocol

In Fig. 3, h_{zkp} is assumed to return an element of $\mathbb{Z}_q$. The reader should observe in Fig. 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, v_2, 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 $<\mathsf{svr}, pk_{\mathsf{svr}}, \tau, t, v_0, v_1, a>$, 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 Fig. 4.

The overall goal of the protocol in Fig. 4 is to generate a new share x_1' for server svr′, and new ticket τ' for the device to use with svr′. x_1' is constructed in a fashion similar to how d_1' was constructed in Fig. 2, specifically as $x_1' \equiv_q x_{10}' + x_{11}' \equiv_q (x_0 - x_0' + \Delta) + (x_1 - \Delta)$, with the first and second terms being computed by dvc and svr, respectively. In Fig. 4, v_2, β, γ, δ, δ' and α play roles similar to those in S-RSA-DEL. It is important that the device deletes v_2, β, b', ρ and all other intermediate results when the protocol completes, and it never stores them on stable storage.

For reasons similar to those in Sect. 3.4, to achieve provable security it is necessary that dvc complete the protocol in Fig. 4 only if $\pi = \pi_0$; otherwise, dvc must abort the protocol. As there is nothing in the protocol of Fig. 4 that itself enables dvc to confirm that $\pi = \pi_0$, this must be confirmed somehow outside the protocol, but without storing additional state on dvc that might permit an offline dictionary attack to occur if dvc is captured. Similar to our approach in Sect. 3.4, we recommend that dvc concurrently execute the decryption protocol in Fig. 3 on a ciphertext c formed by dvc by, e.g., encrypting a newly generated random message. If that protocol successfully decrypts c to reveal the same message, then $\pi = \pi_0$ with high probability and dvc is cleared to successfully complete the protocol of Fig. 4.

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 Changing the password

As described previously, a primary motivation for our protocols, and specifically property G2, is to render the user's password (and thus her device's private key) largely invulnerable to dictionary attacks in the event that her device is stolen. In practice, however, the user's password may be disclosed in other ways, e.g., by being observed. In this case, the user may wish to change her password, so that if the attacker subsequently captures her device, she will be afforded the stronger protections of G2 versus only G4. In this section we informally describe how our S-RSA-DEL and D-ELG-DEL systems can be adopted to support password changes.

In examining the protocols of the previous sections, the password π is an input in the computation of two values, namely the value $b = f_1(v_1, \pi)$ in each ticket τ (and the corresponding β in requests to the server accompanying τ) and the user's share of the private key, i.e., either $d_0 = f_0(v_0, \pi)$

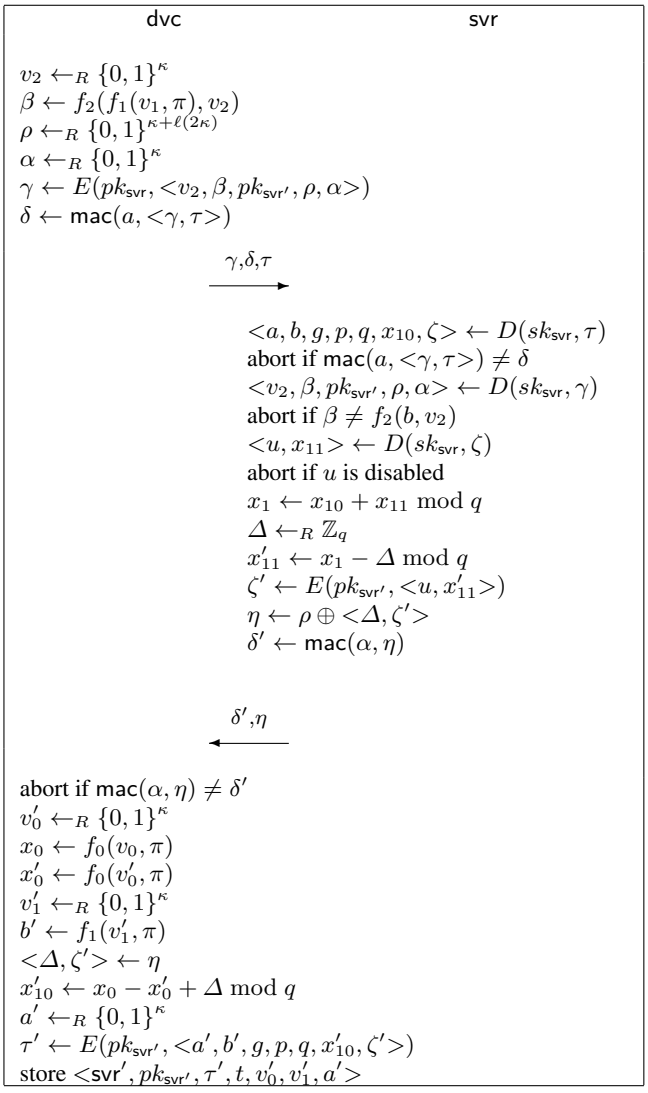

Fig. 4. D-ELG-DEL delegation protocol

in S-RSA-DEL or $x_0 = f_0(v_0,\pi)$ in D-ELG-DEL. Therefore, a strategy for changing the user's password from π to π' is to execute a variation of the delegation protocol to adjust b and the user's share for use with a server. In the case of S-RSA-DEL, this new password-changing protocol would be identical to that of Fig. 2 except that d'_0 would be computed as $d'_0 \leftarrow f_0(v'_0,\pi')$, and b' would be computed as $b' \leftarrow f_1(v'_1,\pi')$. In this way, the ticket τ' would then be consistent with the new password π', and would not work with the old password π.

After running this password-changing protocol, the device would need to delete the old authorization record reflecting π, and the password change would be complete at this server. In order to change her password globally, for each authorized server svr, either this password-changing protocol would need to be executed with svr, or else svr would need to be revoked. For simplicity, we do not consider this extension further in this paper, or more specifically, in the correctness proofs of Sects. 6 and 7.

6 Security for S-RSA-DEL

In this section we provide a formal proof of security for the S-RSA-DEL system, as described in Sect. 3, in the random oracle model. We begin, however, with some intuition for the goals G1–G4 in light of the protocols of Figs. 1 and 2.

- An A1 attacker never obtains an authorization record $<\mathsf{svr}, pk_{\mathsf{svr}}, \tau, t, v_0, v_1, a>$ from dvc, either because it never compromises dvc or because dvc has deleted all such records by the time it is compromised. Without any v_0 used with any svr, the attacker has no ability to forge a signature for dvc (even if it knows π_0); this is property G1.
- An A2 attacker *can* obtain $<\mathsf{svr}, pk_{\mathsf{svr}}, \tau, t, v_0, v_1, a>$ for some svr, but only for a svr that is never compromised. Thus, the attacker has no information about the d_1 in τ (or more precisely, the values d_{10} and ζ encoded in τ, or the value d_{11} encoded in ζ) and can forge only by succeeding in an online dictionary attack with some such svr (goal G2).
- For an A3 attacker, since some $\mathsf{svr} \in \mathsf{Active}(\mathsf{dvc.comp})$ is compromised, the attacker can obtain its d_{10} and ζ by decrypting τ for svr, and then d_{11} by decrypting ζ. The A3 attacker can then conduct an offline dictionary attack on π_0 using the v_0 for svr and $d_1 \leftarrow d_{10} + d_{11}$, and so goal G3 is the best that can be achieved in this case.
- An attacker in class A4 compromises both π_0 and dvc when there is at least one active svr (and so it learns d_0 for svr). Moreover, the attacker can delegate from svr to any other $\mathsf{svr}' \in U$, and obviously will learn the d'_0 for that svr′. Thus, to achieve disabling, it is necessary that the attacker never corrupts any svr (and so never learns any d_1 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.

6.1 Definitions

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

Security for pseudorandom functions. A bit $\xi \leftarrow_R \{0,1\}$ is selected randomly. If $\xi = 0$, then the attacker A is given access to an oracle for $f(v)$ where $v \leftarrow_R \{0,1\}^\kappa$. If $\xi = 1$, then A is given access to an oracle for a random function with the same domain and range as $f(v)$. A can query its oracle with arbitrary elements in the domain and receive its outputs. Finally, A must choose $\xi' \in \{0,1\}$ and succeeds if $\xi' = \xi$. We say that A (q,ϵ)-breaks the pseudorandom function if the attacker makes q queries to the oracle, and $\Pr(A \text{ succeeds}) \geq \frac{1}{2} + \epsilon$. For simplicity, we assume in the proofs below that ϵ is always negligible as a function of κ. As a result, if v remains unknown to the attacker, then $f(v)$ can be replaced with a random function, and the attacker will have negligible probability of noticing a difference.

Security for encryption schemes. We specify adaptive chosen-ciphertext security [20] 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 $\xi \leftarrow_R \{0,1\}$, and returns $Y \leftarrow E(pk, X_\xi)$. Then A continues as before, with the one restriction that it cannot query the decryption oracle on Y. Finally A outputs ξ', and succeeds if $\xi' = \xi$. 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$[\mathcal{E}, \mathcal{D}, \mathcal{M}]$ denote an S-RSA-DEL system based on encryption scheme $\mathcal{E}$, dictionary $\mathcal{D}$, and probabilistic *mistype function* $\mathcal{M} : \mathcal{D} \rightarrow \mathcal{D}$. $\mathcal{M}$ models the ways in which users mistype passwords, and is assumed to satisfy $\forall \pi \in \mathcal{D} : \Pr[\mathcal{M}(\pi) = \pi] = 0$. As described in Sect. 2.2.1, in our proof of S-RSA-DEL, we will require the forger to select when dvc is operated prior to dvc.comp with an incorrect password (see startSign and startDel below). When this is requested, $\mathcal{M}$ determines what password is used instead.

A forger is given $<e, N>$ where ($<e, N>$, $<d, N, \phi(N)>$) $\leftarrow G_{\mathrm{RSA}}(1^\lambda)$, 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 handleSign(γ, δ, τ) 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 handleDel(γ, δ, τ) 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 disable(t) query the svr oracle rejects all future queries with tickets containing ticket identifiers equal to $h_{\mathsf{dsbl}}(t)$ (see Sect. 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. A startSign(m, svr, χ) query represents a request to initiate the S-RSA-DEL signature protocol with either the correct password (if $\chi = 1$) or with an incorrect password (if $\chi = 0$). 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 finishSign(η) 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. A startDel$(\mathsf{svr}, \mathsf{svr}', \chi)$ query represents a request to initiate the S-RSA-DEL delegation protocol with either the correct (if $\chi = 1$) or an incorrect (if $\chi = 0$) password. If svr is authorized, the dvc oracle returns an output message γ, δ, τ, and sets some internal state. On the corresponding finishDel(δ', η) 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′. Recall that as stipulated in Sect. 3.4, if finishDel(δ', η) corresponds to a startDel$(\mathsf{svr}, \mathsf{svr}', 0)$ query, then dvc aborts. On a revoke(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 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 handleSign(γ, δ, τ) query, where $D(sk_{\mathsf{svr}}, \gamma) = <m, *, *, *, *>$, before all servers received 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 3, where we model f_0, f_1, and f_2 as random oracles, let q_{f_0}, q_{f_1}, and q_{f_2} be the number of queries to the respective random oracles. Let q_o be the number of other oracle queries not counted above. Let $\overline{q} = (q_{\mathsf{dvc}}, q_{\mathsf{svr}}, q_o, q_{f_0}, q_{f_1}, q_{f_2})$. In a slight abuse of notation, let $|\overline{q}| = q_{\mathsf{dvc}} + q_{\mathsf{svr}} + q_o + q_{f_0} + q_{f_1} + q_{f_2}$, 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.

6.2 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 Sect. 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 that f_0, f_1 and f_2 are random oracles, and that $f_0(v_0)$ (for random v_0) and $f_1(v_1)$ (for random v_1) have negligible probabilities of collision over $\mathcal{D}$. (The latter property would be achieved, for example, if f_0 and f_1 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 example, to prove security against a forger in class A2, we require that $\{f_0(v)\}_{v\in\{0,1\}^\kappa}$, $\{f_1(v)\}_{v\in\{0,1\}^\kappa}$ and $\{f_2(v)\}_{v\in\{0,1\}^\kappa}$ are pseudorandom function families, and that $f_1(v)$ (for random v) has a negligible probability of collision over $\mathcal{D}$. For proving security against a class A4 forger we make no requirement on f_0, f_1, or f_2.

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 1. *Suppose that $\{f_0(v)\}_{v\in\{0,1\}^\kappa}$ is a pseudorandom function family. If a class A1 forger $(\overline{q}, \epsilon)$-breaks the* S-RSA-DEL$[\mathcal{E}, \mathcal{D}, \mathcal{M}]$ *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. Let Real$'$ denote the S-RSA-DEL system in which all instances of $f_0(v_0)$ are replaced by perfectly random functions, and suppose that F forges in Real$'$ with probability ϵ'. Then, by the pseudorandomness of $\{f_0(v)\}_{v\in\{0,1\}^\kappa}$, $\epsilon' \approx \epsilon$. Now we construct a forger F^* for the RSA signature scheme that receives an RSA public key $<e, N>$ and corresponding signature oracle as input, and runs a simulation of Real$'$ for F. Whenever F forges in this simulation, F^* will forge an RSA signature.

F^* runs the following simulation of the Real$'$ 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 $<\mathsf{svr}_0, pk_{\mathsf{svr}_0}, \tau, t, v_0, v_1, a>$ as normal, except setting $d_1 \leftarrow_R \mathbb{Z}_N$. F^* stores this authorization record and this corresponding value of d_1; d_1 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 Real$'$.
- F^* responds to a startSign(m, svr, χ) query by querying the signature oracle to get σ and r (or deriving r from σ, if necessary), and responding as in Real$'$ using that r value and, if $\chi = 0$, using $\pi \leftarrow \mathcal{M}(\pi_0)$ in place of π_0. F^* responds to a finishSign(η) query corresponding to a startSign$(m, \mathsf{svr}, 0)$ query by simulating a dvc abort. F^* responds to a finishSign(η) query corresponding to a startSign$(m, \mathsf{svr}, 1)$ query by looking up the authorization record $<\mathsf{svr}, pk_{\mathsf{svr}}, \tau, t, v_0, v_1, a>$ and the correct server share d_1 for τ, computing $\nu \leftarrow \eta \oplus \rho$ (for the ρ value from the startSign(m, svr) query) and checking that $\nu \equiv_N (\mathsf{encode}(m, r))^{d_1}$. If this is false, F^* simulates a dvc abort. Otherwise, F^* returns $<\sigma, r>$.
- F^* responds to startDel$(\mathsf{svr}, \mathsf{svr}', \chi)$ as in Real$'$, except using $\pi \leftarrow \mathcal{M}(\pi_0)$ in place of π_0 if $\chi = 0$. F^* responds to a finishDel(δ', η) query corresponding to a startDel$(\mathsf{svr}, \mathsf{svr}', 0)$ query by simulating a dvc abort. F^* responds to a finishDel(δ', η) query corresponding to a startDel$(\mathsf{svr}, \mathsf{svr}', 1)$ query by looking up the authorization record $<\mathsf{svr}, pk_{\mathsf{svr}}, \tau, t, v_0, v_1, a>$ and the correct server share d_1 for τ, behaving as in Real$'$ to generate and store a new authorization record $<\mathsf{svr}', pk_{\mathsf{svr}'}, \tau', t, v_0', v_1', a'>$, and storing $d_1' \leftarrow d_1 + d_0 - d_0'$ as the correct server share for τ'. F^* responds to revoke queries as in Real$'$.

From F's perspective, the above simulation is statistically indistinguishable from Real$'$. (It is not perfectly indistinguishable from Real$'$ due to the choice of d_1 in initialization, and the possibility that F^* simulates a dvc abort when this would not have happened in Real$'$ due to the F guessing the output of a random function $f_0(v_0)$.) So, if F forges with probility ϵ' in Real$'$, then it forges with probability $\epsilon'' \approx \epsilon' \approx \epsilon$ 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 2. *Suppose that $\{f_0(v)\}_{v\in\{0,1\}^\kappa}$, $\{f_1(v)\}_{v\in\{0,1\}^\kappa}$, and $\{f_2(v)\}_{v\in\{0,1\}^\kappa}$ are pseudorandom function families, and that $f_1(v)$ (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$[\mathcal{E}, \mathcal{D}, \mathcal{M}]$ *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.) Consider the following systems, each building upon the next:

- Let Real$_1$ be the S-RSA-DEL system in which all instances of $f_0(v_0)$ used for servers that are not good are replaced by perfectly random functions. If F forges in Real$_1$ with probability $\frac{q_{\mathsf{svr}}}{|\mathcal{D}|} + \psi_1$, then by pseudorandomness of f_0 we know that $\psi_1 \approx \psi$.
- Let Real$_2$ be the Real$_1$ system in which all instances of $f_1(v_1)$ used for servers that are not good are replaced by perfectly random functions. If F forges in Real$_2$ with probability $\frac{q_{\mathsf{svr}}}{|\mathcal{D}|} + \psi_2$, then by pseudorandomness of f_1 we know that $\psi_2 \approx \psi_1$.
- Let Real$_3$ be the Real$_2$ system with values B_1, ..., $B_{q_{\mathsf{dvc}}} \leftarrow_R \{0,1\}^\kappa$ selected during initialization, and so that if the i-th dvc query is of the form startSign($*$, svr, 0) or startDel(svr, $*$, 0), utilizes $\pi \leftarrow \mathcal{M}(\pi_0)$, and svr is not good, then $f_1(v_1, \pi)$ is set to B_i (if $f_1(v_1, \pi)$ was not previously set to some B_j, $j < i$). Since each $f_1(v_1)$ is a random function, $f_1(v_1, \pi)$ is never disclosed to F,[2] and $B_1, \ldots, B_{q_{\mathsf{dvc}}}$ are chosen randomly, F forges in Real$_3$ with probability $\frac{q_{\mathsf{svr}}}{|\mathcal{D}|} + \psi_3$ with $\psi_3 = \psi_2$.

[2] Recall from Sect. 3.4 that dvc must abort any delegation using a $\pi \neq \pi_0$ before completion, and so no $f_1(v_1, \pi)$ for $\pi \neq \pi_0$ will ever be inserted in a ticket τ by dvc.

- Let Real_4 be the Real_3 system with functions $f_2(B_1), \ldots, f_2(B_{q_{\mathsf{dvc}}})$ replaced by perfectly random functions. If F forges in Real_4 with probability $\frac{q_{\mathsf{svr}}}{|\mathcal{D}|} + \psi_4$, then by pseudorandomness of f_2 we know that $\psi_4 \approx \psi_3$.
- Let Real_5 be the Real_4 system utilizing a single random function f in place of the random functions $f_2(B_1), \ldots, f_2(B_{q_{\mathsf{dvc}}})$. F distinguishes Real_5 from Real_4 only if the same value v_2 is chosen in two distinct dvc queries of the form startSign($*$, svr, 0) or startDel(svr, $*$, 0) where svr is not good (since $f(v_2)$ would repeat, whereas $f_2(B_i, v_2)$ and $f_2(B_j, v_2)$ may be different). This happens with probability at most $(q_{\mathsf{dvc}})^2/2^{\kappa}$, which is negligible. Therefore, if F forges in Real_5 with probability $\frac{q_{\mathsf{svr}}}{|\mathcal{D}|} + \psi_5$, then $\psi_5 \approx \psi_4$.

Now we construct a simulator Sim for Real_5 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 $\frac{q_{\mathsf{svr}}}{|\mathcal{D}|} + \frac{\psi_5}{2}$, then we will be able to construct a forger that $(q_{\mathsf{dvc}}, \epsilon')$-breaks the RSA signature scheme with $\epsilon' \approx \frac{\psi_5}{2} \approx \frac{\psi}{2}$. If, on the other hand, F wins against Sim with probability at most $\frac{q_{\mathsf{svr}}}{|\mathcal{D}|} + \frac{\psi_5}{2}$, 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 that was stored on the device for svr, v_1 was stored in the authorization record with τ, and either
  - for a handleSign query to svr, γ was not generated by a device startSign($*$, svr, 1) query, $<*, *, v_2, \beta, *> \leftarrow D(sk_{\mathsf{svr}}, \gamma)$, and $\beta = f_2(f_1(v_1, \pi_0), v_2)$,
  - for a handleDel query to svr, γ was not generated by a device startDel(svr, $*$, 1) query, $<v_2, \beta, *, *, *> \leftarrow D(sk_{\mathsf{svr}}, \gamma)$, and $\beta = f_2(f_1(v_1, \pi_0), v_2)$; or
- τ is not a ticket that was stored on the device for svr, γ was generated by a device startSign$(*, \mathsf{svr}, 1)$ or startDel$(\mathsf{svr}, *, 1)$ query using a record $<\mathsf{svr}, *, *, *, *, v_1, *>$, $<*, b', *, *, *> \leftarrow D(sk_{\mathsf{svr}}, \tau)$, and $b' = f_1(v_1, \pi_0)$.

We now define Sim. Below, we say that Sim *zeroes* a ticket τ for svr if Sim generates $\tau \leftarrow E(pk_{\mathsf{svr}}, 0^{3\kappa+\ell(2\kappa+\lambda)+2\lambda})$, and we call the values $<a, b, d_{10}, \zeta, N>$ present when Sim creates τ the *zeroed inputs* to τ. Similarly, Sim zeroes a value γ for svr if Sim generates $\gamma \leftarrow E(pk_{\mathsf{svr}}, 0^{|m|+\kappa_{sig}+2\kappa+\lambda})$ (respectively, $\gamma \leftarrow E(pk_{\mathsf{svr}}, 0^{4\kappa+2\lambda+\ell(2\kappa+\lambda)})$) in a startSign($m$, svr, $*$) (resp., startDel(svr, svr′, $*$)) oracle query, and the values $<m, r, v_2, \beta, \rho>$ (resp., $<v_2, \beta, pk_{\mathsf{svr}'}, \rho, \alpha>$) 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_1 is chosen as $d_1 \leftarrow_R \mathbb{Z}_N$. 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_1 is called the "correct server share" for τ.

Sim responds to oracle queries as follows (using truly random functions for f_0 and f_2 when used with non-good servers, as in Real_5).

- 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 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 handleSign(γ, δ, τ) or handleDel(γ, δ, τ) 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 startSign(m, svr, χ) query as in Theorem 1, except if svr is good, it zeroes γ. Sim responds to a dvc finishSign(η) query as in Theorem 1.
- Sim responds to a dvc startDel$(\mathsf{svr}, \mathsf{svr}', \chi)$ query as in Theorem 1, except if svr is good, it zeroes γ. Sim responds to a dvc finishDel(δ', η) query corresponding to a startDel$(\mathsf{svr}, \mathsf{svr}', \chi)$ query as in Theorem 1, except if svr′ is good, it zeroes τ'.

Suppose that the probability of F winning against Sim is more than $\frac{q_{\mathsf{svr}}}{|\mathcal{D}|} + \frac{\psi_5}{2}$. Since f_0 and f_2 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 $f_1(v_1)$ over $\mathcal{D}$ for one of q_{dvc} random v_1'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 $\frac{q_{\mathsf{svr}}}{|\mathcal{D}|} + \frac{\psi_5}{2}$. Since F forges in Real_5 with probability $\frac{q_{\mathsf{svr}}}{|\mathcal{D}|} + \psi_5$, it wins in the Real_5 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_5 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_5 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_5 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

$\frac{q_{\mathsf{svr}}}{|\mathcal{D}|} + \frac{\psi_5}{2}$ 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 $\frac{q_{\mathsf{svr}}}{|\mathcal{D}|} + \psi_5$.

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 $\approx \frac{\psi_5}{2(2q_{\mathsf{dvc}}+1)}$. 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 $\epsilon'' \approx \frac{\psi_5}{2|U|(2q_{\mathsf{dvc}}+1)} \approx \frac{\psi}{2|U|(2q_{\mathsf{dvc}}+1)}$.

Our next proof is of property G3, i.e., for an A3 attacker. In order to prove security against this attacker, however, we have to make additional assumptions about $\mathcal{M}$. To understand why, note that an A3 attacker, who performs both dvc.comp and svr.comp for some svr that is active when dvc.comp occurs, learns v_0 and v_1 used for svr. Consequently, and unlike an A2 attacker, an A3 attacker *can* obtain $f_1(v_1, \pi)$ for passwords $\pi \neq \pi_0$: it simply uses v_1 and evaluates $f_1(v_1, \pi)$ for any password π of its choosing. If dvc had previously revealed a v_2, β pair where $\beta = f_2(f_1(v_1, \pi), v_2)$ and $\pi \neq \pi_0$ – as it would in response to a dvc $\mathsf{startSign}(*, \mathsf{svr}, 0)$ query using $\pi \leftarrow \mathcal{M}(\pi_0)$ – the attacker can then confirm a guess at π. Depending on the properties of $\mathcal{M}$, relatively few guesses at π can significantly reduce the adversary's effort to find π_0 (the only secret remaining in the system). For example, suppose there is an even partition $\mathcal{D}_1, \mathcal{D}_2$ of $\mathcal{D}$ (i.e., $|\mathcal{D}_1| = |\mathcal{D}_2|$, $\mathcal{D}_1 \cap \mathcal{D}_2 = \emptyset$, $\mathcal{D}_1 \cup \mathcal{D}_2 = \mathcal{D}$) and elements $\pi_1, \pi_2 \in \mathcal{D}$ such that $\forall \pi \in \mathcal{D}_1 : \mathcal{M}(\pi) = \pi_1$ and $\forall \pi \in \mathcal{D}_2 : \mathcal{M}(\pi) = \pi_2$ (deterministically). Then the adversary need only confirm whether π_1 or π_2 was used in the construction of a v_2, β pair to reduce the search for π_0 by a factor of two! In order to state a theorem for an A3 attacker, then, we stipulate that $\mathcal{M}$ be *diffuse* in the following sense: $\mathcal{M}$ is *diffuse* if

$$\forall \pi \in \mathcal{D} : \Pr[\pi_0 \leftarrow_R \mathcal{D}; \pi \leftarrow \mathcal{M}(\pi_0)] = 1/|\mathcal{D}|. \quad (2)$$

Intuitively, $\mathcal{M}$ is diffuse if it spreads mistypes of different passwords evenly across the password space. This seems to be a reasonable assumption, since presumably the mistypes of a password depend on the password that the user is attempting to type. With this definition, we now state and prove Theorem 3.

Theorem 3. *Suppose f_0, f_1, and f_2 are random oracles, and that $f_0(v_0)$ (for random v_0) and $f_1(v_1)$ (for random v_1) have negligible probabilities of collision over $\mathcal{D}$. Also suppose that $\mathcal{M}$ is diffuse. If a class A3 forger $(\overline{q}, \epsilon)$-breaks the* S-RSA-DEL$[\mathcal{E}, \mathcal{D}, \mathcal{M}]$ *system with $\epsilon = \frac{q_{f_0}+q_{f_1}+q_{f_2}}{|\mathcal{D}|} + \psi$, 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 $\frac{q_{f_0}+q_{f_1}+q_{f_2}}{|\mathcal{D}|} + \psi$. 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 1, except that the simulation aborts if the adversary *nearly guesses the password*. Here, the adversary *nearly guesses the password* if it either (i) queries the f_0 oracle or f_1 oracle with π_0 in its second argument, i.e., it actually guesses the password and confirms it; or (ii) queries $f_1(v_1, \pi)$ to obtain some value b, where v_1 and π are used in a $\mathsf{startSign}(*, *, 0)$ or $\mathsf{startDel}(*, *, 0)$ query (i.e., $\pi \leftarrow \mathcal{M}(\pi_0)$), and also queries $f_2(b, v_2)$ for the value v_2 generated in that $\mathsf{startSign}(*, *, 0)$ or $\mathsf{startDel}(*, *, 0)$ query. Since $\mathcal{M}$ is diffuse, (ii) occurs with probability at most $\frac{q_{f_2}}{|\mathcal{D}|} + \frac{(q_{f_1})^2}{2^\kappa}$ (with the second term coming from the possibility of a collision in $f_1(v_1)$), and so the total probability of F nearly guessing the password is at most negligibly greater than $\frac{q_{f_0}+q_{f_1}+q_{f_2}}{|\mathcal{D}|}$.

This simulation is statistically indistinguishable from the real system (from F's viewpoint) unless the simulation aborts, the probability of which is the probability of F nearly guessing the password. So since F forges with probability $\frac{q_{f_0}+q_{f_1}+q_{f_2}}{|\mathcal{D}|} + \psi$ 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 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$[\mathcal{E}, \mathcal{D}, \mathcal{M}]$ *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 $d_1 \leftarrow 0$ and $\zeta \leftarrow E(pk_{\mathsf{svr}_0}, 0^{\lambda+2\kappa})$. Sim then saves the authorization record $<\mathsf{svr}_0, pk_{\mathsf{svr}_0}, \tau, t, v_0, v_1, a>$. For the ciphertext ζ, the tuple $<pk_{\mathsf{svr}_0}, \zeta, d_0>$ is recorded on an *offset list*.[3] Sim then responds to oracle queries from F as follows:

- Sim responds to dvc oracle queries as in the real system, except using $\pi \leftarrow \mathcal{M}(\pi_0)$ in place of π_0 in $\mathsf{startSign}(*, *, 0)$ and $\mathsf{startDel}(*, *, 0)$ queries and their corresponding $\mathsf{finishSign}(*)$ and $\mathsf{finishDel}(*, *)$ queries.
- 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 $<a$,

[3] The third value in this tuple is an overestimate of the value of the device share d_0, including the value d_{10} that is added to create the server share d_1. For the original ticket τ, this d_{10} value is actually zero, and thus the third value in the tuple is the true value of the device share d_0.

$b, d_{10}, \zeta, N> \leftarrow D(sk_{\mathsf{svr}}, \tau)$ and $<v_2, \beta, pk_{\mathsf{svr}'}, \rho, \alpha> \leftarrow D(sk_{\mathsf{svr}}, \gamma)$. Now Sim examines the offset list with pk_{svr} and ζ. If for some d_0^+, $<pk_{\mathsf{svr}}, \zeta, d_0^+>$ 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 $d_0 \leftarrow d_0^+ - d_{10}$,
  - computes $\Delta \leftarrow \{0,1\}^{\lambda+\kappa}$ (i.e., as normal), and
  - stores $<pk_{\mathsf{svr}'}, \zeta', d_0 + \Delta>$ 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 $<a, b, d_{10}, \zeta, N> \leftarrow D(sk_{\mathsf{svr}}, \tau)$ and $<m, r, v_2, \beta, \rho> \leftarrow D(sk_{\mathsf{svr}}, \gamma)$. Now Sim examines the offset list with pk_{svr} and ζ. If for some d_0^+, $<pk_{\mathsf{svr}}, \zeta, d_0^+>$ appears in the offset list, then Sim causes svr to abort if u is marked as disabled at svr, and otherwise
  - computes $d_0 \leftarrow d_0^+ - d_{10}$,
  - 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 $\nu \leftarrow \sigma(\mathsf{encode}(m, r')^{-d_0}) \bmod N$.

  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 $(<e, N>, <d, N, \phi(N)>) \leftarrow G_{\mathrm{RSA}}(1^\lambda)$ 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, t, u, v_0, v_1, 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 $X_0 = <u, d_1>$ and $X_1 = 0^{\kappa+\lambda}$ and records the tuple $<pk_{\mathsf{svr}_0}, \zeta, d_1>$ on an *offset list*.[4] If $i > 0$ or $\mathsf{svr}_0 \neq \mathsf{svr}^*$, then it sets ζ normally and records the tuple $<pk_{\mathsf{svr}_0}, \zeta, d_1>$ on the offset list. After this, A proceeds normally to complete the authorization record $<\mathsf{svr}_0, pk_{\mathsf{svr}_0}, \tau, t, v_0, v_1, a>$. A then simulates S-RSA-DEL for F as follows.

- A responds to dvc oracle queries as in the real system, except using $\pi \leftarrow \mathcal{M}(\pi_0)$ in place of π_0 in startSign$(*, *, 0)$ and startDel$(*, *, 0)$ queries and their corresponding finishSign$(*)$ and finishDel$(*, *)$ queries.
- A responds to a svr handleDel(γ, δ, τ) 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 handleDel query (to any server) in which the server did not abort. At this point A has computed $<a, b, d_{10}, \zeta, N> \leftarrow D(sk_{\mathsf{svr}}, \tau)$ and $<v_2, \beta, pk_{\mathsf{svr}'}, \rho, \alpha> \leftarrow D(sk_{\mathsf{svr}}, \gamma)$ (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 $<u', d_{11}>$, and if $pk = pk_{\mathsf{svr}}$ then A calls the decryption oracle on ζ to obtain $<u', d_{11}>$. 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_{11}, $<pk_{\mathsf{svr}}, \zeta, d_{11}>$ 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 $d_1 \leftarrow d_{10} + d_{11}$, computes Δ and d'_{11} as normal, and alters the computation of ζ' as follows. If $j < i$, then A computes $\zeta' \leftarrow E(pk_{\mathsf{svr}'}, <u, d'_{11}>)$ as normal and records $<pk_{\mathsf{svr}'}, \zeta', d'_{11}>$ in the offset list. If $j = i$ and $\mathsf{svr}' = \mathsf{svr}^*$, then A calls the test oracle for pk with $X_0 = <u, d'_{11}>$ and $X_1 = 0^{\lambda+2\kappa}$, obtains the response ζ', and records $<pk_{\mathsf{svr}'}, \zeta', d'_{11}>$ 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 $<pk_{\mathsf{svr}'}, \zeta', d'_{11}>$ in the offset list. In all cases, A then computes η and δ' normally and responds.
- A responds to a svr handleSign(γ, δ, τ) 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 $<a, b, d_{10}, \zeta, N> \leftarrow D(sk_{\mathsf{svr}}, \tau)$ and $<m, r, v_2, \beta, \rho> \leftarrow D(sk_{\mathsf{svr}}, \gamma)$ (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 $<u', d_{11}>$, and if $pk = pk_{\mathsf{svr}}$ then A calls the decryption oracle on ζ to obtain $<u', d_{11}>$. 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_{11}, $<pk_{\mathsf{svr}}, \zeta, d_{11}>$ 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 $d_1 \leftarrow d_{10} + d_{11}$, sets $\nu \leftarrow \mathsf{encode}(m, r)^{d_1} \bmod N$, 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.

[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 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 $\frac{\epsilon}{2|U|(q_{\mathsf{svr}}+1)}$.

7 Security for D-ELG-DEL

In this section we provide a formal proof of security for the D-ELG-DEL system, as described in Sect. 4 and instantiated with an ElGamal-like encryption scheme. Specifically, we provide a formal proof of goals G1–G4.

7.1 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. Assume that the public/private key pair is

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

and that the decryption oracle receives a ciphertext c. If $\mathsf{valid}(c) = 0$, it returns $\perp$. Otherwise, it returns $z = (\mathsf{select}(c))^x \bmod p$, from which the decryption of c can be computed using $\mathsf{reveal}(z, c)$. (Note that in the standard definition of security for encryption schemes, the decryption oracle would not return z, but the decryption of c.)

Note: The TDH1 and TDH2 encryption schemes from [22], when restricted to a single server, are in fact ELGL encryption schemes secure in 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$[\mathcal{E}, \mathcal{D}, \mathcal{M}]$ denote an D-ELG-DEL system based on an encryption scheme $\mathcal{E}$, dictionary $\mathcal{D}$, and probabilistic *mistype function* $\mathcal{M} : \mathcal{D} \to \mathcal{D}$. $\mathcal{M}$ models the ways in which users mistype passwords, and is assumed to satisfy $\forall \pi \in \mathcal{D} : \Pr[\mathcal{M}(\pi) = \pi] = 0$. As described in Sect. 2.2.1, in our proof of D-ELG-DEL, we will require the attacker to select when dvc is operated prior to dvc.comp with an incorrect password (see startDecr and startDel below). When this is requested, $\mathcal{M}$ determines what password is used instead.

An attacker A is given $<g, p, q, y>$ where $(<g, p, q, y>, <g, p, q, x>) \leftarrow G_{\mathrm{ElG}}(1^\lambda)$, and 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 Sect. 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 $\mathsf{startDecr}(c, \mathsf{svr}, \chi)$ query, which represents a request to initiate the D-ELG-DEL decryption protocol with either the correct password (if $\chi = 1$) or an incorrect password (if $\chi = 0$), 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 startDecr query. On a $\mathsf{startDel}(\mathsf{svr}, \mathsf{svr}', \chi)$ 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'. Recall that as stipulated in Sect. 4.4, if $\mathsf{finishDel}(\delta', \eta)$ corresponds to a $\mathsf{startDel}(\mathsf{svr}, \mathsf{svr}', 0)$ query, then dvc aborts. 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, the attacker A generates two equal length strings X_0 and X_1 and sends these to the test oracle, which chooses $\xi \leftarrow_R \{0, 1\}$, and returns the ELGL encryption $Y \leftarrow E(pk, X_\xi)$. Then A continues as before, with the restriction that if A is of class A1, A2, or A3, 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 ξ', and succeeds if $\xi' = \xi$.

Let q_{dvc}, q_{svr}, q_{f_0}, q_{f_1}, and q_{f_2} be defined in the same manner as in Sect. 6. Let q_o be the number of other oracle queries not counted above. Let $\overline{q} = (q_{\mathsf{dvc}}, q_{\mathsf{svr}}, q_o, q_{f_0}, q_{f_1}, q_{f_2})$. In a slight abuse of notation, let $|\overline{q}| = q_{\mathsf{dvc}} + q_{\mathsf{svr}} + q_o + q_{f_0} + q_{f_1} + q_{f_2}$, 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 $\Pr(A \text{ succeeds}) \geq \frac{1}{2} + \epsilon$.

7.2 Theorems

Here we prove that if an attacker breaks the D-ELG-DEL $[\mathcal{E}, \mathcal{D}, \mathcal{M}]$ 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 Sect. 6, we prove security separately for the different classes of attackers from Sect. 2.2.

Theorem 5. *Let h_{zkp} be a random oracle and let $\{f_0(v)\}_{v\in\{0,1\}^\kappa}$ be a pseudorandom function family. If a class A1 attacker $(\overline{q}, \epsilon)$-breaks the* D-ELG-DEL$[\mathcal{E}, \mathcal{D}, \mathcal{M}]$ *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. Let Real' denote the D-ELG-DEL system in which all instances of $f_0(v_0)$ are replaced by perfectly random functions, and suppose that A breaks Real' with probability ϵ'. Then, by the pseudorandomness of $\{f_0(v)\}_{v\in\{0,1\}^\kappa}$, $\epsilon' \approx \epsilon$. Now we 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 Real' 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 $<\mathsf{svr}_0, pk_{\mathsf{svr}_0}, \tau, t, v_0, v_1, a>$ as normal, except setting $x_1 \leftarrow_R \mathbb{Z}_q$. A^* stores this authorization record and this corresponding value of x_1; x_1 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 Real'.
- 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 Y is the response.
- A^* responds to h_{zkp} queries as a normal random oracle.
- A^* responds to a $\mathsf{startDecr}(c, \mathsf{svr}, \chi)$ query as in Real', though using $\pi \leftarrow \mathcal{M}(\pi_0)$ if $\chi = 0$. A^* responds to a $\mathsf{finishDecr}(\eta, e, s)$ query corresponding to a $\mathsf{startDecr}(c, \mathsf{svr}, 0)$ query by simulating a dvc abort. A^* responds to a $\mathsf{finishDecr}(\eta, e, s)$ query corresponding to a $\mathsf{startDecr}(c, \mathsf{svr}, 1)$ query as follows. (Without loss of generality, we may assume $\mathsf{valid}(c) = 1$ and $c \neq Y$.) A^* computes $w \leftarrow \mathsf{select}(c)$. Then A^* looks up the authorization record $<\mathsf{svr}, pk_{\mathsf{svr}}, \tau, t, v_0, v_1, a>$ and the correct server share x_1 for τ and computes $\nu \leftarrow \eta \oplus \rho$ (for the ρ value from the $\mathsf{startDecr}(c, \mathsf{svr}, 1)$ query). If $\nu^q \not\equiv_p 1$ or $e \neq h_{\mathsf{zkp}}(<w, \nu, w^s\nu^{-e} \bmod p, g^{s-ex_1} \bmod p>)$ then A^* simulates a dvc abort. Subsequently, if $\nu \not\equiv_p w^{x_1}$, then A^* itself aborts and outputs 0. Otherwise, A^* queries the ELGL decryption oracle with c to obtain a value z and computes the plaintext $m \leftarrow \mathsf{reveal}(z, c)$, which is then returned by A^* in response to the $\mathsf{finishDecr}(\eta, e, s)$ query.
- A^* responds to revoke queries as in Real'. A^* responds to a $\mathsf{startDel}(\mathsf{svr}, \mathsf{svr}', \chi)$ as in the real system, except using $\pi \leftarrow \mathcal{M}(\pi_0)$ in place of π_0 if $\chi = 0$. A^* responds to a $\mathsf{finishDel}(\delta', \eta)$ query corresponding to a $\mathsf{startDel}(\mathsf{svr}, \mathsf{svr}', 0)$ query by simulating a dvc abort. A^* responds to a $\mathsf{finishDel}(\delta', \eta)$ query corresponding to a $\mathsf{startDel}(\mathsf{svr}, \mathsf{svr}', 1)$ query by looking up the authorization record $<\mathsf{svr}, pk_{\mathsf{svr}}, \tau, t, v_0, v_1, a>$ and the correct server share x_1 for τ, behaving as in Real' to generate a new authorization record $<\mathsf{svr}', pk_{\mathsf{svr}'}, \tau', t, v_0', v_1', a'>$, and storing $x_1' \leftarrow x_1 + x_0 - x_0' \bmod q$ as the correct server share for τ'.

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

Now, to analyze the simulation, note that if A^* itself does not abort, then the response by the simulation for a device $\mathsf{finishDecr}(\eta, e, s)$ query corresponding to a $\mathsf{startDecr}(c, \mathsf{svr}, 1)$ query is exactly the same as the response in Real', since the z value computed by A^* would be the same as the z value computed by the device. Here we show that the probability that A^* aborts is negligible. First note that if $\nu^q \equiv_p 1$ and $\nu \not\equiv_p w^{x_1}$, then for any values ν' and ν'', the probability that there is an s for which $e = h_{\mathsf{zkp}}(<w, \nu, \nu', \nu''>)$, $\nu' \equiv_p w^s\nu^{-e}$, and $\nu'' \equiv_p g^{s-ex_1}$ is $1/q$. Thus the probability that A ever produces values η, e, s that determine such values ν, ν', ν'' is negligible.

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

Theorem 6. *Suppose that h_{zkp} is a random oracle, that $\{f_0(v)\}_{v\in\{0,1\}^\kappa}$, $\{f_1(v)\}_{v\in\{0,1\}^\kappa}$ and $\{f_2(v)\}_{v\in\{0,1\}^\kappa}$ are pseudorandom function families, and that $f_1(v)$ (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$[\mathcal{E}, \mathcal{D}, \mathcal{M}]$ *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$[\mathcal{E}, \mathcal{D}, \mathcal{M}]$ 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.) Consider the following systems, each building upon the next:

- Let Real_1 be the D-ELG-DEL system in which all instances of $f_0(v_0)$ used for servers that are not good are replaced by perfectly random functions. If A succeeds in the Real_1 system with probability $\frac{1}{2} + \frac{q_{\mathsf{svr}}}{|\mathcal{D}|} + \psi_1$, then by the pseudorandomness of f_0 we know that $\psi_1 \approx \psi$.
- Let Real_2 be the Real_1 system in which all instances of $f_1(v_1)$ used for servers that are not good are replaced by perfectly random functions. If A succeeds in Real_2 with probability $\frac{1}{2} + \frac{q_{\mathsf{svr}}}{|\mathcal{D}|} + \psi_2$, then by pseudorandomness of f_1 we know that $\psi_2 \approx \psi_1$.
- Let Real_3 be the Real_2 system with values $B_1, \ldots, B_{q_{\mathsf{dvc}}} \leftarrow_R \{0,1\}^\kappa$ selected during initialization, and so that if the i-th dvc query is of the form startDecr($*$, svr, 0) or startDel(svr, $*$, 0), utilizes $\pi \leftarrow \mathcal{M}(\pi_0)$, and svr is not good, then $f_1(v_1, \pi)$ is set to B_i (if $f_1(v_1, \pi)$ was not previously set to some B_j, $j < i$). Since each $f_1(v_1)$ is a random function, $f_1(v_1, \pi)$ is never disclosed to A, and $B_1, \ldots, B_{q_{\mathsf{dvc}}}$ are chosen randomly, A succeeds in Real_3 with probability $\frac{1}{2} + \frac{q_{\mathsf{svr}}}{|\mathcal{D}|} + \psi_3$ with $\psi_3 = \psi_2$.
- Let Real_4 be the Real_3 system with functions $f_2(B_1), \ldots, f_2(B_{q_{\mathsf{dvc}}})$ replaced by perfectly random functions. If A succeeds in Real_4 with probability $\frac{1}{2} + \frac{q_{\mathsf{svr}}}{|\mathcal{D}|} + \psi_4$, then by pseudorandomness of f_2 we know that $\psi_4 \approx \psi_3$.
- Let Real_5 be the Real_4 system utilizing a single random function f in place of the random functions $f_2(B_1), \ldots, f_2(B_{q_{\mathsf{dvc}}})$. A distinguishes Real_5 from Real_4 only if the same value v_2 is chosen in two distinct dvc queries of the form startDecr($*$, svr, 0) or startDel(svr, $*$, 0) where svr is not good (since $f(v_2)$ would repeat, whereas $f_2(B_i, v_2)$ and $f_2(B_j, v_2)$ may be different). This happens with probability at most $(q_{\mathsf{dvc}})^2/2^\kappa$, which is negligible. Therefore, if A succeeds in Real_5 with probability $\frac{1}{2} + \frac{q_{\mathsf{svr}}}{|\mathcal{D}|} + \psi_5$, then $\psi_5 \approx \psi_4$.

Now we construct a simulator Sim for Real_5 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 $\frac{1}{2} + \frac{q_{\mathsf{svr}}}{|\mathcal{D}|} + \frac{\psi_5}{2}$, then we will be able to construct an attacker that $(q_{\mathsf{dvc}}, \epsilon'')$-breaks the underlying ELGL scheme with $\epsilon'' \approx \frac{\psi_5}{2} \approx \frac{\psi}{2}$. If, on the other hand, A wins against Sim with probability at most $\frac{1}{2} + \frac{q_{\mathsf{svr}}}{|\mathcal{D}|} + \frac{\psi_5}{2}$, 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 ξ 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 that was stored on the device for svr, v_1 was stored in the authorization record with τ, and either
  - for a handleDecr query to svr, γ was not generated by a device startDecr($*$, svr, 1) query, $<*, v_2, \beta, *> \leftarrow D(sk_{\mathsf{svr}}, \gamma)$, and $\beta = f_2(f_1(v_1, \pi_0), v_2)$,
  - for a handleDel query to svr, γ was not generated by a device startDel(svr, $*$, 1) query, $<v_2, \beta, *, *, *> \leftarrow D(sk_{\mathsf{svr}}, \gamma)$, and $\beta = f_2(f_1(v_1, \pi_0), v_2)$; or
- τ is not a ticket that was stored on the device for svr, γ was generated by a device startDecr($*$, svr, 1) or startDel(svr, $*$, 1) query using a record $<$svr, $*$, $*$, $*$, $*$, v_1, $*>$, $<*, b', *, *, *, *, *> \leftarrow D(sk_{\mathsf{svr}}, \tau)$, and $b' = f_1(v_1, \pi_0)$.

We now define Sim. Below, we say that Sim *zeroes* a ticket τ for svr if Sim generates $\tau \leftarrow E(pk_{\mathsf{svr}}, 0^{4\kappa+2\lambda+\ell(2\kappa)})$, and we call the values $<a, b, g, p, q, x_{10}, \zeta>$ present when Sim creates τ the *zeroed inputs* to τ. Similarly, Sim zeroes a value γ for svr if Sim generates $\gamma \leftarrow E(pk_{\mathsf{svr}}, 0^{|c|+2\kappa+\lambda})$ (respectively, $\gamma \leftarrow E(pk_{\mathsf{svr}}, 0^{4\kappa+\lambda+\ell(2\kappa)})$) in a startDecr($c$, svr, $*$) (resp., startDel(svr, svr$'$, $*$)) oracle query, and the values $<c, v_2, \beta, \rho>$ (resp., $<v_2, \beta, pk_{\mathsf{svr}'}, \rho, \alpha>$) 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 A. 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_1 is chosen as $x_1 \leftarrow_R \mathbb{Z}_q$. 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_1 is called the "correct server share" for τ.

Sim responds to oracle queries as follows (using truly random functions for f_0 and f_2 when used with non-good servers, as in Real_5).

- Sim responds to queries to the test oracle as in Theorem 5.
- Sim responds h_{zkp} queries as a normal random oracle.
- Sim responds to a dvc.comp query by giving all authorization records stored on the device to A. Sim responds to a svr.comp query by giving sk_{svr} to A.
- Sim responds to a svr 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 handleDecr(γ, δ, τ) or handleDel(γ, δ, τ) 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 handleDecr 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 startDecr(c, svr, χ) query as in Theorem 5, except if svr is good, it zeroes γ. Sim responds to a dvc finishDecr(η, e, s) query as in Theorem 5.
- Sim responds to a dvc startDel(svr, svr$'$, χ) query as in Theorem 5, except if svr is good, it zeroes γ. Sim responds to a dvc finishDel(δ', η) query corresponding to a startDel(svr, svr$'$, χ) query as in Theorem 5, except if svr$'$ is good it zeroes τ'.

Suppose that the probability of A winning against Sim is more than $\frac{1}{2} + \frac{q_{\mathsf{svr}}}{|\mathcal{D}|} + \frac{\psi_5}{2}$. Since f_0 and f_2 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 $f_1(v_1)$ over $\mathcal{D}$ for one of q_{dvc} random v_1's. So, A succeeds in guessing the bit ξ chosen by the test oracle with probability $\frac{1}{2} + \epsilon''$ where $\epsilon'' \approx \frac{\psi_5}{2} \approx \frac{\psi}{2}$. An attacker A^{**} for the the underlying ELGL scheme can thus run Sim for A and output the bit ξ 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 $\frac{1}{2} + \frac{q_{\mathsf{svr}}}{|\mathcal{D}|} + \frac{\psi_5}{2}$. Since A succeeds in Real_5 with probability $\frac{1}{2} + \frac{q_{\mathsf{svr}}}{|\mathcal{D}|} + \psi_5$, it wins in Real_5 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_5 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_5 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_5 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 $\frac{1}{2} + \frac{q_{\mathsf{svr}}}{|\mathcal{D}|} + \frac{\psi_5}{2}$ 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 $\frac{1}{2} + \frac{q_{\mathsf{svr}}}{|\mathcal{D}|} + \psi_5$.

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 $\approx \frac{\psi_5}{2(2q_{\mathsf{dvc}}+1)}$. 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 $(i+1)^{\mathrm{st}}$ 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 $\epsilon' \approx \frac{\psi_5}{2|U|(2q_{\mathsf{dvc}}+1)} \approx \frac{\psi}{2|U|(2q_{\mathsf{dvc}}+1)}$.

In order to prove property G3, we need to require that $\mathcal{M}$ be *diffuse* – see (2) in Sect. 6 – as we did for Theorem 3 in the case of S-RSA-DEL.

Theorem 7. *Let f_0, f_1, f_2 and h_{zkp} be random oracles. Also suppose that $f_0(v_0)$ (for random v_0) and $f_1(v_1)$ (for random v_1) have negligible probabilities of collision over $\mathcal{D}$, and that $\mathcal{M}$ is diffuse. If a class A3 attacker $(\overline{q}, \epsilon)$-breaks the* D-ELG-DEL$[\mathcal{E}, \mathcal{D}, \mathcal{M}]$ *system with $\epsilon = \frac{q_{f_0}+q_{f_1}+q_{f_2}}{|\mathcal{D}|} + \psi$, 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$[\mathcal{E}, \mathcal{D}, \mathcal{M}]$ system with $\epsilon = \frac{q_{f_0}+q_{f_1}+q_{f_2}}{|\mathcal{D}|} + \psi$. 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, except that the simulation aborts if the adversary *nearly guesses the password*. Here, the adversary *nearly guesses the password* if it either (i) queries the f_0 or f_1 oracle with π_0 in its second argument, i.e., it actually guesses the password and confirms it; or (ii) queries $f_1(v_1, \pi)$ to obtain some value b, where v_1 and π are used in a $\mathsf{startDecr}(*, *, 0)$ or $\mathsf{startDel}(*, *, 0)$ query (i.e., $\pi \leftarrow \mathcal{M}(\pi_0)$), and also queries $f_2(b, v_2)$ for the value v_2 generated in that $\mathsf{startDecr}(*, *, 0)$ or $\mathsf{startDel}(*, *, 0)$ query. Since $\mathcal{M}$ is diffuse, (ii) occurs with probability at most $\frac{q_{f_2}}{|\mathcal{D}|} + \frac{(q_{f_1})^2}{2^\kappa}$ (with the second term coming from the possibility of a collision in $f_1(v_1)$), and so the total probability of A nearly guessing the password is at most negligibly greater than $\frac{q_{f_0}+q_{f_1}+q_{f_2}}{|\mathcal{D}|}$.

This simulation is statistically indistinguishable from the real system (from A's viewpoint) unless the simulation aborts, the probability of which is exactly that of A nearly guessing the password. So since A succeeds with probability $\frac{1}{2} + \epsilon$ in the real system with $\epsilon = \frac{q_{f_0}+q_{f_1}+q_{f_2}}{|\mathcal{D}|} + \psi$, A succeeds with probability at least $\frac{1}{2} + \epsilon'$ in the simulation, where $\epsilon' \approx \psi$. If A outputs a bit ξ for the D-ELG-DEL system, then A^* outputs the same bit ξ for the underlying ELGL scheme and thereby $(q_{\mathsf{dvc}}, \epsilon')$-breaks the underlying ELGL scheme with $\epsilon' \approx \psi$.

Theorem 8. *Suppose that h_{zkp} is a random oracle. If a class A4 attacker $(\overline{q}, \epsilon)$-breaks the* D-ELG-DEL$[\mathcal{E}, \mathcal{D}, \mathcal{M}]$ *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 $x_1 \leftarrow 0$ and $\zeta \leftarrow E(pk_{\mathsf{svr}_0}, 0^{2^\kappa})$. Sim then saves the authorization record $<\mathsf{svr}_0, pk_{\mathsf{svr}_0}, \tau, t, v_0, v_1, a>$. For the ciphertext ζ, the tuple $<pk_{\mathsf{svr}_0}, \zeta, x_0>$ 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 5.
- Sim responds to dvc oracle queries as in the real system, except using $\pi \leftarrow \mathcal{M}(\pi_0)$ in place of π_0 in $\mathsf{startDecr}(*, *, 0)$ and $\mathsf{startDel}(*, *, 0)$ queries and their corresponding $\mathsf{finishDecr}(*, *, *)$ and $\mathsf{finishDel}(*, *)$ queries.

- 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 $<a, b, g, p, q, x_{10}, \zeta> \leftarrow D(sk_{\mathsf{svr}}, \tau)$ and $<v_2, \beta, pk_{\mathsf{svr}'}, \rho, \alpha> \leftarrow D(sk_{\mathsf{svr}}, \gamma)$. Now Sim examines the offset list with pk_{svr} and ζ. If for some x_0^+, $<pk_{\mathsf{svr}}, \zeta, x_0^+>$ 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 $x_0 \leftarrow x_0^+ - x_{10} \bmod q$,
  - computes $\Delta \leftarrow \mathbb{Z}_q$ (i.e., as normal), and
  - stores $<pk_{\mathsf{svr}'}, \zeta', x_0 + \Delta \bmod q>$ 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 handleDecr(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 $<a, b, g, p, q, x_{10}, \zeta> \leftarrow D(sk_{\mathsf{svr}}, \tau)$ and $<c, v_2, \beta, \rho> \leftarrow D(sk_{\mathsf{svr}}, \gamma)$. Now Sim examines the offset list with pk_{svr} and ζ. If for some x_0^+, $<pk_{\mathsf{svr}}, \zeta, x_0^+>$ appears in the offset list, then Sim causes svr to abort if u is marked as disabled at svr, and otherwise
  - computes $x_0 \leftarrow x_0^+ - x_{10} \bmod q$,
  - queries the decryption oracle on c to obtain z,
  - computes $\nu \leftarrow zw^{-x_0} \bmod p$,
  - generates $e \leftarrow_R \mathbb{Z}_q$ and $s \leftarrow_R \mathbb{Z}_q$, and
  - backpatches h_{zkp}:
    $h_{\mathsf{zkp}}(<w, \nu, w^s \nu^{-e} \bmod p,$
    $g^s (y g^{-x_0})^{-e} \bmod p>) \leftarrow e$

  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 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 ξ 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 $(<g, p, q, y>, <g, p, q, x>) \leftarrow G_{\mathrm{ElG}}(1^\lambda)$ 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, t, u, v_0, v_1, 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 $X_0 = <u, x_1>$ and $X_1 = 0^{2\kappa}$ and records the tuple $<pk_{\mathsf{svr}_0}, \zeta, x_1>$ on an *offset list*.[5] If $i > 0$ or $\mathsf{svr}_0 \neq \mathsf{svr}^*$, then it sets ζ normally and records the tuple $<pk_{\mathsf{svr}_0}, \zeta, x_1>$ on the offset list. After this, A^* proceeds normally to complete the authorization record $<\mathsf{svr}_0, pk_{\mathsf{svr}_0}, \tau, t, v_0, v_1, a>$. A^* then simulates D-ElG-Del for A as follows.

- A^* responds to queries to the test oracle as in Theorem 5.
- A^* responds to dvc oracle queries as in the real system, except using $\pi \leftarrow \mathcal{M}(\pi_0)$ in place of π_0 in startDecr$(*, *, 0)$ and startDel$(*, *, 0)$ queries and their corresponding finishDecr$(*, *, *)$ and finishDel$(*, *)$ queries.
- A^* responds to a svr handleDel(γ, δ, τ) 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 handleDel query (to any server) in which the server did not abort. At this point A has computed $<a, b, g, p, q, x_{10}, \zeta> \leftarrow D(sk_{\mathsf{svr}}, \tau)$ and $<v_2, \beta, pk_{\mathsf{svr}'}, \rho, \alpha> \leftarrow D(sk_{\mathsf{svr}}, \gamma)$ (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 $<u', x_{11}>$, and if $pk = pk_{\mathsf{svr}}$ then A^* calls the decryption oracle on ζ to obtain $<u', x_{11}>$. 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_{11}, $<pk_{\mathsf{svr}}, \zeta, x_{11}>$ 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 $x_1 \leftarrow x_{10} + x_{11}$, computes Δ and x'_{11} as normal, and alters the computation of ζ' as follows. If $j < i$, then A^* computes $\zeta' \leftarrow E(pk_{\mathsf{svr}'}, <u, x'_{11}>)$ as normal and records $<pk_{\mathsf{svr}'}, \zeta', x'_{11}>$ in the offset list. If $j = i$ and $\mathsf{svr}' = \mathsf{svr}^*$, then A calls the test oracle for pk with $X_0 = <u, x'_{11}>$ and $X_1 = 0^{2\kappa}$, obtains the response ζ', and records $<pk_{\mathsf{svr}'}, \zeta', x'_{11}>$ 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 $<pk_{\mathsf{svr}'}, \zeta', x'_{11}>$ in the offset list. In all cases, A^* then computes η and δ' normally and responds.
- A^* responds to a svr handleDecr(γ, δ, τ) 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 $<a, b, g, p, q, x_{10}, \zeta> \leftarrow D(sk_{\mathsf{svr}}, \tau)$ and $<c, v_2, \beta, \rho> \leftarrow D(sk_{\mathsf{svr}}, \gamma)$ (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 $<u', x_{11}>$, and if $pk = pk_{\mathsf{svr}}$ then A^* calls the decryption oracle on ζ to obtain $<u', x_{11}>$. If u' is marked as disabled at

[5] 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).

svr, then A^* simulates a svr abort, and otherwise A^* continues as svr would normally.
  - If for some x_{11}, $<pk_{\mathsf{svr}}, \zeta, x_{11}>$ 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 $x_1 \leftarrow x_{10} + x_{11} \bmod q$, 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 the bit ξ 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, P. Rogaway. Relations among notions of security for public-key encryption schemes. In *Advances in Cryptology – CRYPTO '98* (Lecture Notes in Computer Science 1462), pp. 26–45, 1998
2. M. Bellare, P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In 1^{st} *ACM Conf. on Comp. and Comm. Security*, pp. 62–73, 1993
3. M. Bellare, P. Rogaway. Optimal asymmetric encryption. In *Advances in Cryptology – EUROCRYPT '94* (Lecture Notes in Computer Science 950), pp. 92–111, 1995
4. M. Bellare, P. Rogaway. The exact security of digital signatures – How to sign with RSA and Rabin. In *Advances in Cryptology – EUROCRYPT '96* (Lecture Notes in Computer Science 1070), pp. 399–416, 1996
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, V. Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *Advances in Cryptology – CRYPTO '98* (Lecture Notes in Computer Science 1462), pp. 13–25, 1998
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, 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, 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, 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. D. W. Kravitz. Digital signature algorithm. U.S. Patent 5,231,668, 27 July 1993
15. P. MacKenzie, 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
16. P. MacKenzie, M. K. Reiter. Two-party generation of DSA signatures. In *Advances in Cryptologie – CRYPTO 2001* (Lecture Notes in Computer Science 2139), pp. 138–154, August 2001
17. P. MacKenzie, M. K. Reiter. Delegation of cryptographic servers for capture-resilient devices (extended abstract). In 8^{th} *ACM Conf. on Comp. and Comm. Security*, Nov. 2001
18. R. Morris, K. Thompson. Password security: A case history. *Comm. of the ACM* **22**(11), 594–597 (Nov. 1979.)
19. U. Maurer, S. Wolf. The Diffie-Hellman protocol. *Designs, Codes, and Cryptography* **19**, 147–171, Kluwer Academic Publishers, 2000
20. C. Rackoff, D. Simon. Noninteractive zero-knowledge proof of knowledge and chosen ciphertext attack. In *Advances in Cryptology – CRYPTO '91*, (Lecture Notes in Computer Science 576), pp. 433–444, 1991
21. R. L. Rivest, A. Shamir, L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Comm. of the ACM* **21**(2), 120–126 (Feb. 1978)
22. V. Shoup, R. Gennaro. Securing threshold cryptosystems against chosen ciphertext attack. In *Advances in Cryptology – EUROCRYPT '98* (Lecture Notes in Computer Science 1403), pp. 1–16, 1998

Philip MacKenzie is a Member of Technical Staff at Bell Laboratories, the research and development arm of Lucent Technologies. He received a B.Sc. degree in computer science and mathematics from the University of Michigan in 1987, and received M.Sc. and Ph.D. degrees in computer science in 1988 and 1992, respectively, also at the University of Michigan. He continued on with postdoctoral work at the University of Texas in 1992, and then at Sandia National Laboratories in 1994. He joined Boise State University in 1996 as an Assistant Professor in the Mathematics and Computer Science Department, before joining Bell Labs in 1998. His research interests include cryptography and all areas of computer and communications security.

Michael K. Reiter is a Professor of Electrical & Computer Engineering and Computer Science at Carnegie Mellon University in Pittsburgh, Pennsylvania, USA. He received the B.Sc. degree in mathematical sciences from the University of North Carolina in 1989, and the M.Sc. and Ph.D. degrees in computer science from Cornell University in 1991 and 1993, respectively. He joined AT&T Bell Labs in 1993 and became a founding member of AT&T Labs – Research when NCR and Lucent Technologies (including Bell Labs) were split away from AT&T in 1996. He returned to Bell Labs in 1998 as Director of Secure Systems Research, and then joined Carnegie Mellon in 2001. His research interests include all areas of computer and communications security and distributed computing.