Intrusion-Resilient Public-Key Encryption

Yevgeniy Dodis[1], Matt Franklin[2], Jonathan Katz[3], Atsuko Miyaji[2,4], and Moti Yung[5]

[1] Department of Computer Science, New York University. dodis@cs.nyu.edu
[2] Department of Computer Science, University of California, Davis. {miyaji,franklin}@cs.ucdavis.edu
[3] Department of Computer Science, University of Maryland, College Park. jkatz@cs.umd.edu
[4] Japan Advanced Institute of Science and Technology. miyaji@jaist.ac.jp
[5] Department of Computer Science, Columbia University. moti@cs.columbia.edu

Abstract. Exposure of secret keys seems to be inevitable, and may in practice represent the most likely point of failure in a cryptographic system. Recently, the notion of *intrusion-resilience* [17] (which extends both the notions of *forward security* [3, 5] and *key insulation* [11]) was proposed as a means of mitigating the harmful effects that key exposure can have. In this model, time is divided into distinct periods; the public key remains fixed throughout the lifetime of the protocol but the secret key is periodically updated. Secret information is stored by both a *user* and a *base*; the user performs all cryptographic operations during a given time period, while the base helps the user periodically update his key. Intrusion-resilient schemes remain secure in the face of multiple compromises of both the user and the base, as long as they are not both compromised simultaneously. Furthermore, in case the user and base *are* compromised simultaneously, prior time periods remain secure (as in forward-secure schemes).

Intrusion-resilient signature schemes have been previously constructed [17, 15]. Here, we give the first construction of an intrusion-resilient public-key encryption scheme, based on the recently-constructed forward-secure encryption scheme of [8]. We also consider generic transformations for securing intrusion-resilient encryption schemes against chosen-ciphertext attacks.

1 Introduction

Exposure of secret keys is perhaps the most debilitating attack on a cryptosystem since it typically implies that all security guarantees are lost. This problem is emerging as an ever-greater threat as cryptographic primitives are deployed on inexpensive, lightweight, and mobile devices; in these cases, it is typically much easier for an adversary to break into the device and obtain the secret keys than to crack the computational assumptions on which the system is based. Clearly, concerns about key exposure must be addressed in a satisfactory manner by the research community.

Recognizing the need to address these concerns, a long line of research has focused on dealing with the threat of key exposure. Methods to prevent key exposure entirely (e.g., by using tamper-resistant devices) seem cost-prohibitive and impractical for most common applications. Thus, research has focused on making key exposures more difficult, or, alternately, minimizing the damage when (partial) key exposure occurs. As an example, threshold cryptography [10, 9] distributes secrets among n devices so that exposure of secrets from, say, t of these devices will not allow an adversary to "break" the scheme. On the other hand, this requires that at least $t + 1$ devices participate every time a cryptographic operation must be performed. While this may be acceptable in some scenarios, this does not seem appropriate for mobile users and in other settings where the risk of key exposure is high but users need the ability to perform cryptographic computations on their own.

Alternative approaches to this problem have been proposed whereby the key required for cryptographic computations always resides on a single device. In such proposals, time is divided into distinct periods $1, \ldots, N$ and secret keys *evolve* over time (public keys, however, are fixed for the lifetime of the scheme). The goal here is to contain — as much as possible — damage from key exposures that occur. *Forward-secure cryptosystems* [3, 5] were the first solutions in this vein. In forward-secure schemes, the secret key is stored by a single user and this key is updated by the user at the beginning of every time period. An adversary who exposes a key for period t can perform cryptographic operations (signing, decrypting, etc.) for periods $t' \geq t$ but cannot break the scheme (in the appropriate sense) for any prior time periods $t' < t$. The effect of key exposure is thereby contained as much as possible given that a single user stores all secret keying information.

To address the issue of obtaining security for time periods *following* key exposure, the notion of *key-insulation* [11, 12] was proposed. This model had the distinguishing feature of assuming (a limited amount of) secure storage on a server with which the user periodically interacts. (Here, one can imagine the "user" as a mobile device and the "server" as a desktop PC in the user's home.) The user can perform all cryptographic operations during any particular time period on his own and can also update his keys — with the help of the server — at discrete time intervals as above. Because a limited amount of secure storage is assumed, the security obtainable here is better than in the forward-secure case; in particular, schemes can be designed such that an adversary who exposes keys stored by the user multiple times (i.e., at time periods $T = \{t_1, \ldots, t_\ell\}$) cannot "break" the scheme for *any* other time period either in the past or in the future (i.e., for any time periods $t \notin T$). In *strong* key-insulated schemes, exposing only the secret keys stored on the server does not permit an adversary to "break" the scheme at all.

Recently, these models were synthesized into the most powerful notion set forth to date: *intrusion-resilience* [17]. As in the key-insulated model, this model assumes a user who performs all cryptographic operations and a server with which the user interacts to update his keys at discrete time intervals. Now, however, it is no longer assumed that the server is secure. Since key exposures at the server are now assumed to occur frequently, the user and server have the option of "refreshing" their secrets (this is reminiscent of proactivation [23]). Here (informally speaking; see the formal definition in Section 2), schemes can be designed such that an adversary who exposes keys stored at *both* the user and the server on multiple occasions — but never at the same time — cannot "break" the scheme for any time periods other than those for which keys were exposed at the user. Furthermore, in case the keys of the user and server *are* both exposed at some time t (and no refresh was performed in between these exposures), the scheme remains forward-secure so that the adversary cannot "break" the scheme at any prior time periods $t' < t$.

We note that each of these models may be appropriate in different environments. Forward-secure schemes are advantageous in that the user is self-sufficient and need not interact with any other device. On the other hand, the security provided by key-insulated and intrusion-resilient schemes is better and these schemes might therefore be used when interacting with a server is feasible and does not represent a serious drawback. Finally, although the intrusion-resilient model offers stronger security guarantees than the key-insulated model, we note that solutions for the latter are (thus far) much more efficient. The choice of which type of scheme to use therefore depends heavily on an assumption about the (physical) security of the server.

1.1 Our Contributions

Much work has focused on the design and analysis of forward-secure signature schemes [5, 20, 2, 16, 21] and, more recently, a forward-secure public-key encryption scheme has been constructed [8]. Key insulated public-key encryption schemes [11, 7, 6] and signature schemes [12] are also known. Thus far, however, only constructions of intrusion-resilient signature schemes [17, 15] have been proposed.[6] Here, we give the first definitions of intrusion-resilient public-key encryption and the first construction of an intrusion-resilient public-key encryption scheme. Our scheme is based on the recent forward-secure encryption scheme of [8], and the security of our scheme is therefore based on the BDH assumption in the random oracle model. As in [8], we may modify our scheme so as to achieve semantic security in the *standard model* under the *decisional* BDH assumption.

We also consider generic transformations for securing intrusion-resilient public-key encryption schemes against adaptive chosen-ciphertext attacks (in the random oracle model). We show that *any* such transformation that works for "standard" public-key encryption schemes also works for intrusion-resilient public-key encryption schemes.[7] In particular, then, we may apply known conversions (e.g., those of [13, 22]) to our scheme so as to obtain the first intrusion-resilient public-key encryption scheme secure against chosen-ciphertext attacks.

2 Definitions and Preliminaries

The definitions given here are the first to appear for the case of intrusion-resilient encryption; they exactly parallel those appearing in [17] for the case of intrusion-resilient signatures.

In our model time is divided into distinct periods labeled $1, \ldots, N$. We have a *base* and a *user* who (jointly) establish a public key which remains fixed for the duration of the protocol. Encryption of a message depends on the current time period; thus, ciphertexts have the form $\langle t, C\rangle$ where t indicates the time period during which encryption was performed. The base and the user each store secret keying information: at time period t, the user stores sufficient information to decrypt messages sent during time t (so, to decrypt, the user does not need to contact the base). At the beginning of time period $t+1$, the base updates its keys and sends an *update message* to the user; this update message, in particular, will contain information enabling the user to compute the key needed to decrypt during period $t+1$. Finally, the base can also send *refresh messages* to the

[6] As mentioned by [17], forward-secure public-key encryption is necessary in order to build intrusion-resilient public-key encryption. At that time, however, no non-trivial forward-secure public-key encryption schemes were known.

[7] As pointed out in [8], this does *not* seem to be the case for forward-secure encryption schemes.

user at any point in time; if a refresh is executed between compromises of the user and the base, the system remains secure (refreshes thus serve a similar purpose to proactivation [23]).

As in [17], the adversary in our model can obtain secrets from the base and the user for multiple, adaptively-chosen time periods. The adversary can also intercept update and refresh messages sent by the base. Informally speaking (a precise definition is given below), if the adversary only compromises the base, the encryption scheme remains secure. If the adversary compromises the user repeatedly, the encryption scheme remains secure except for periods whose secrets were obtained (either directly or by combination of compromise and interception of update/refresh messages). Finally, even if the base and the user are compromised during the same time period (and no refresh was sent in the interim), the encryption scheme remains secure for prior time periods.

2.1 Functional Specification

We first define correct operation of the scheme, and defer a definition of security to the next section. The notation $SK_{t,r}$ (respectively, $SKB_{t.r}$) denotes the user's (respectively, base's) secret key for time period t following r refreshes. We say $t.r = t'.r'$ if $t = t'$ and $r = r'$. We say $t.r < t'.r'$ if $t < t'$ or if $t = t'$ and $r < r'$. As in prior work, we assume for convenience that a key update occurs immediately after key generation to obtain keys for $t = 1$, and that a key refresh occurs immediately after every key update to obtain keys for $r = 1$.

Definition 1. *A key-updating public-key encryption scheme Π consists of the following* PPT *algorithms:*

- Gen*, the key generation algorithm, takes as input security parameter 1^k and the total number of time periods N. It outputs the initial user key $SK_{0.0}$, the initial base key $SKB_{0.0}$, and the public key PK.*
- Enc*, the encryption algorithm, takes as input the public key PK, a time period t, and a message m. It outputs ciphertext $\langle t, C\rangle$. (Note that refreshes are transparent to the sender, but updates are not since they occur at well-defined intervals.)*
- Dec*, the decryption algorithm, takes as input the current user key $SK_{t.r}$ and a ciphertext $\langle t, C\rangle$. It outputs a message m. (As usual, we require that, for any r, we have $\mathsf{Dec}(SK_{t.r}, \mathsf{Enc}(PK, t, m)) = m$.)*
- $\mathsf{UpdBase}$*, the base key update algorithm, takes as input the current base key $SKB_{t.r}$. It outputs a new base key $SKB_{t+1.0}$ for the next time period as well as key update message SKU_t.*

- UpdUser, *the user key update algorithm, takes as input the current user key* $SK_{t.r}$ *and a key update message* SKU_t*. It outputs the new user key* $SK_{t+1.0}$ *for the next time period.*
- RefBase, *the base key refresh algorithm, takes as input the current base key* $SKB_{t.r}$*. It outputs a new base key* $SKB_{t.r+1}$ *as well as key refresh message* $SKR_{t.r}$*.*
- RefUser, *the user key refresh algorithm, takes as input the current user key* $SK_{t.r}$ *and a key refresh message* $SKR_{t.r}$*. It outputs a new user key* $SK_{t.r+1}$*.*

2.2 Definition of Security

To aid our definition of security, we let $RN(t)$ denote the number of refreshes that occur in time period t. Recall that each update is assumed to be followed by an immediate refresh, so keys with $r = 0$ are never actually used. RN is for notational convenience only; it need not be known in advance.

Consider the following "thought experiment" which generates all keying information for the entire lifetime of the scheme:

Experiment Generate-Keys(k, N, RN)
  $t := 0, r := 0$
  $(SK_{0.0}, SKB_{0.0}, PK) \leftarrow \mathsf{Gen}(1^k, N)$
  for $t = 1$ to N
    $(SKB_{t.0}, SKU_{t-1}) \leftarrow \mathsf{UpdBase}(SKB_{(t-1).r})$
    $SK_{t.0} \leftarrow \mathsf{UpdUser}(SK_{(t-1).r}, SKU_{t-1})$
    for $r = 1$ to $RN(t)$
      $(SKB_{t.r}, SKR_{t.(r-1)}) \leftarrow \mathsf{RefBase}(SKB_{t.(r-1)})$
      $SK_{t.r} \leftarrow \mathsf{RefUser}(SK_{t.(r-1)}, SKR_{t.(r-1)})$

Let SK^*, SKB^*, SKU^*, and SKR^* denote the sets of user keys, base keys, update messages, and refresh messages generated in the course of the above experiment. We now define the following oracles available to the adversary:

- LR, the *left-or-right* encryption oracle. On input (t, m_0, m_1) the oracle chooses a random bit b and returns $\mathsf{Enc}(PK, t, m_b)$. This oracle may be queried only once[8] by the adversary, and allows us to define a notion of security.

[8] A hybrid argument (as in [4]) shows that security under a single access to LR is equivalent to security under polynomially-many accesses to LR. We thus use the simpler definition for convenience.

– Osec, the *key exposure* oracle (based on the sets SK^*, SKB^*, SKU^*, and SKR^*) which:
  1. On input (“s”, $t.r$) outputs $SK_{t.r}$;
  2. On input (“b”, $t.r$) outputs $SKB_{t.r}$;
  3. On input (“u”, t) outputs SKU_t and $SKR_{(t+1).0}$;
  4. On input (“r”, $t.r$) outputs $SKR_{t.r}$.

  Queries to this oracle correspond to compromise of the user or base, or to intercepting update or refresh messages. (We assume that queries to this oracle always have t, r within the appropriate bounds.)

For any set Q of key exposure queries, we say that $SK_{t.r}$ is Q-*exposed* if at least one of the following is true:

– (“s”, $t.r$) $\in Q$
– $r > 1$, (“r”, $t.(r-1)$) $\in Q$, and $SK_{t.(r-1)}$ is Q-exposed
– $r = 1$, (“u”,$t-1$) $\in Q$, and $SK_{(t-1).RN(t-1)}$ is Q-exposed

A completely analogous definition may be given for Q-exposure of a base key $SKB_{t.r}$

Clearly, if $SK_{t.r}$ is Q-exposed then the adversary can decrypt messages encrypted during time period t. Similarly, if $SK_{t.r}$ and $SKB_{t.r}$ are both Q-exposed (for some t, r) then the adversary can run the update algorithms itself and thereby decrypt messages encrypted during any time period $t' \geq t$. Thus, we say the scheme is (t, Q)-*compromised* if either $SK_{t.r}$ is Q-exposed (for some r) or if both $SK_{t'.r}$ and $SKB_{t'.r}$ are Q-exposed (for some r and $t' < t$).

We say the adversary *succeeds* if it can determine the bit b used by the LR oracle, where this oracle was queried on a time period t for which the scheme was not (t, Q)-compromised. More formally, consider the following experiment executed with some algorithm A:

Experiment $\mathsf{Run\text{-}Adversary}(A, k, N, RN)$
  $\mathsf{Generate\text{-}Keys}(k, N, RN)$
  $b' \leftarrow A^{\mathsf{LR},\mathsf{Osec}}(1^k, N, PK, RN)$
  Let Q be the set of queries made by A to Osec
  Let (t, m_0, m_1) be the query made to LR
  Let b be the bit used by LR in responding to the query
  if $b \neq b'$ or the scheme is (t, Q)-compromised
    return 0
    else return 1

We define A's probability of success as the probability that 1 is output in the above experiment. The *advantage* of adversary A in attacking scheme

Π (where N and $RN(\cdot)$ are assumed to be clear from context, and are always at most polynomial in k) is defined as twice A's probability of success, minus 1. We denote this advantage by $\mathsf{Adv}_{A,\Pi}(k)$. We may now define security of an intrusion-resilient scheme.

Definition 2. *A key-updating public-key encryption scheme Π is said to be an* intrusion-resilient scheme achieving semantic security *if, for all* PPT *adversaries A and all $N, RN(\cdot)$ polynomial in k, we have $\mathsf{Adv}_{A,\Pi}(k) < \varepsilon(k)$ for some negligible function $\varepsilon(\cdot)$.*

A definition appropriate for describing the concrete security of Π may be easily derived from the above. A definition of security against adaptive chosen-ciphertext attacks is also evident, and we defer such a definition to the full version of this paper.

2.3 Cryptographic Assumptions

The security of our scheme is based on the difficulty of the bilinear Diffie-Hellman (BDH) problem as recently formalized by Boneh and Franklin [7] (see also [19, 18]). We review the relevant definitions as they appear in [7]. Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two cyclic groups of prime order q, where $\mathbb{G}_1$ is represented additively and $\mathbb{G}_2$ is represented multiplicatively. We use a map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ for which the following hold:

1. The map $\hat{e}$ is *bilinear*; that is, for all $P_0, P_1 \in \mathbb{G}_1$ and all $x, y \in \mathbb{Z}_q$ we have $\hat{e}(xP_0, yP_1) = \hat{e}(yP_0, xP_1) = \hat{e}(P_0, P_1)^{xy}$.
2. There is an efficient algorithm to compute $\hat{e}(P_0, P_1)$ for any $P_0, P_1 \in \mathbb{G}_1$.

A *BDH parameter generator $\mathcal{IG}$* is a randomized algorithm that takes a security parameter 1^k, runs in polynomial time, and outputs the description of two groups $\mathbb{G}_1, \mathbb{G}_2$ and a map $\hat{e}$ satisfying the above conditions. We define the *BDH problem with respect to $\mathcal{IG}$* as the following: given $(\mathbb{G}_1, \mathbb{G}_2, \hat{e})$ output by $\mathcal{IG}$ along with random $P, aP, bP, cP \in \mathbb{G}_1$, compute $\hat{e}(P, P)^{abc}$. We say that $\mathcal{IG}$ *satisfies the BDH assumption* if the following is negligible (in k) for all PPT algorithms A:

$$\Pr[(\mathbb{G}_1, \mathbb{G}_2, \hat{e}) \leftarrow \mathcal{IG}(1^k); P \leftarrow \mathbb{G}_1; a, b, c \leftarrow \mathbb{Z}_q : \\ A(\mathbb{G}_1, \mathbb{G}_2, \hat{e}, P, aP, bP, cP) = \hat{e}(P, P)^{abc}].$$

We note that BDH parameter generators for which the BDH assumption is believed to hold can be constructed from Weil and Tate pairings associated with supersingular elliptic curves or abelian varieties. As our results do not depend on any specific instantiation, we refer the interested reader to [7] for details.

3 Construction

Our construction builds on the forward-secure encryption scheme of [8], which is based on previous work of [14] (both of which were enabled by the identity-based encryption scheme of [7]). We assume here that the reader is familiar with the scheme of [8]; in fact, many elements of their scheme are used directly here.

3.1 Scheme Intuition

Assume for simplicity that the total number of time periods N is a power of 2; that is, $N = 2^\ell$. We imagine a full binary tree of height ℓ in which the root is labeled with ε (representing the empty string) and furthermore if a node at depth less than ℓ is labeled with w then its left child is labeled with $w0$ and its right child is labeled with $w1$. Let $\langle t \rangle$ denote the ℓ-bit representation of integer t (where $0 \leq t \leq 2^\ell - 1$). The leaves of the tree (which are labeled with strings of length ℓ) correspond to successive time periods in the obvious way; i.e., time period t is associated with the leaf labeled by $\langle t \rangle$. For simplicity, we refer to the node labeled by w as simply "node w". Every node $w = w_1 \cdots w_j$ in the tree will have an associated "secret point" $S_w \in \mathbb{G}_1$ and all interior nodes also have an associated "translation point" $Q_w \in \mathbb{G}_1$. For all nodes we then have the "local secret key" $sk_w = (S_w, \mathcal{Q}_w)$, where $\mathcal{Q}_w = (Q_{w_1}, \ldots, Q_{w_1 \cdots w_{j-1}})$. We remark that while the translation points are needed for efficient decryption, they do not need to be kept secret.

The properties of these keys will be as follows:

1. To decrypt a message encrypted using PK during period t, only key $sk_{\langle t \rangle}$ is needed.
2. Given key sk_w, it is possible to efficiently derive keys sk_{w0} and sk_{w1}.
3. Given PK and t, and *without* sk_w for all prefixes w of $\langle t \rangle$, it is infeasible to derive $sk_{\langle t \rangle}$ and furthermore infeasible to decrypt messages encrypted during period t.

Once we have a scheme satisfying the above requirements, we utilize the following method. For a given period t, let $t_0 t_1 \cdots t_\ell = \langle t \rangle$, where $t_0 = \epsilon$. The "global secret key" gsk_t for this period will consist of (1) $sk_{\langle t \rangle}$ and also (2) $\{sk_{t_0 t_1 \cdots t_{j-1} 1}\}$ for all $1 \leq j \leq \ell$ such that $t_j = 0$. We denote the latter keys (i.e., the "local secret keys" for all right siblings of nodes on the path from $\langle t \rangle$ to the root) by $\rho(t)$. We refer to the right sibling corresponding to swapping the "last" 0 of $\langle t \rangle$ to 1 as the *deepest sibling*. Also, we let $\mathsf{Sec}_{\langle t \rangle} = (\{S_w \mid w \in \rho(t)\})$ be the set of secret points corresponding to nodes in $\rho(t)$. Since there is some redundant information stored as part of gsk_t (in particular, the translation points do not need to be stored multiple times as part of each local secret key), we may note that gsk_t actually consists of $S_{\langle t\rangle}$, $\mathsf{Sec}_{\langle t\rangle}$, and $\mathcal{Q}_{\langle t\rangle}$.

We may notice that gst_t enables derivation of all the local secret keys for periods t through N (indeed, one can easily derive gsk_{t+1} from gsk_t), but none of the local secret keys for periods $t' < t$. This will allow us to achieve forward security, as in [8]. However, in our model we also need to proactively split gsk_t between the user and the base, so that we can derive the sharing for period $t+1$ from that of period t. To achieve this, we let the user store $sk_{\langle t\rangle}$ — to enable decryption within the current time period — but additively share each secret point in $\mathsf{Sec}_{\langle t\rangle}$ between the user and the base. Intuitively, $sk_{\langle t\rangle}$ by itself only allows the user to decrypt at period t, and the fact that the rest of the global key $\mathsf{Sec}_{\langle t\rangle}$ is split ensures that exposure of the user cannot compromise any of the future periods. Security against compromises of the base is similar (and even simpler since the base does not even store $\mathcal{Q}_{\langle t\rangle}$), except that here, even the current period is secure. Proactivation is simple as well: the base simply randomly refreshes the 2-sharing of $\mathsf{Sec}_{\langle t\rangle}$. This gives us full intrusion-resilience.

The only issue to resolve is how to update (the sharing of) gsk_t to (the sharing of) gsk_{t+1} without compromising security. We notice that this procedure requires choosing several new secret points and translation points. We cannot let the user or the base generate these on their own, since this will compromise the security of the scheme if the corresponding player is corrupted during this phase. Instead, we will have the user and base generate these points *jointly*. The challenge is to do it via a single message from the base to the user. We give a high level description of the main step for doing so. For any node w, if we let $Q_w = s_w P$ for some s_w (where P is some fixed public point) we will have $S_{w0} = S_w + s_w H_1(w0)$ and $S_{w1} = S_w + s_w H_1(w1)$ (here, H_1 is a hash function defined as part of the scheme). Assume the user and base already have a random sharing $S_w = S'_w + S''_w$, and wish to generate Q_w and a random sharing of S_{w0} and S_{w1}. The base chooses a random s'_w, sets $Q'_w = s'_w P$, $S'_{w0} = S'_w + s'_w H_1(w0)$, and $S'_{w1} = S'_w + s'_w H_1(w1)$, and sends Q'_w to the user. The user chooses a random s''_w, sets $Q_w = Q'_w + s''_w P$ (implicitly, $Q_w = (s'_w + s''_w)P$, so $s_w = s'_w + s''_w$), $S''_{w0} = S''_w + s''_w H_1(w0)$, and $S''_{w0} = S''_w + s''_w H_1(w1)$. This step is immediately followed by a random refresh, which ensures that no single party has enough control to cause any security concerns.

3.2 Formal Description

We assume that hash functions $H_1 : \{0,1\}^* \rightarrow \mathbb{G}_1$ and $H_2 : \mathbb{G}_2 \rightarrow \{0,1\}^n$ are defined, either by Gen or else as part of the specification of the scheme. These hash functions will be treated as random oracles in the analysis.

$\mathsf{Gen}(1^k, N = 2^\ell)$ does the following:

1. $\mathcal{IG}(1^k)$ is run to generate group $\mathbb{G}_1, \mathbb{G}_2$ (of order q) and $\hat{e}$.
2. $P \leftarrow \mathbb{G}_1$; $s_\varepsilon \leftarrow \mathbb{Z}_q$. Set $Q = s_\varepsilon P$.
3. The public key is $PK = (\mathbb{G}_1, \mathbb{G}_2, \hat{e}, P, Q)$.
4. Set $S_0 = s_\varepsilon H_1(0)$ and $S_1 = s_\varepsilon H_1(1)$.
5. For $j = 1, \ldots, \ell - 1$:
   (a) $s_{0^j} \leftarrow \mathbb{Z}_q$. Set $Q_{0^j} = s_{0^j} P$.
   (b) Set $S_{0^j 0} = S_{0^j} + s_{0^j} H_1(0^j 0)$ and $S_{0^j 1} = S_{0^j} + s_{0^j} H_1(0^j 1)$.
6. (Note that we have $\mathcal{Q}_{\langle 0 \rangle} = \{Q_0, \ldots, Q_{0^{\ell-1}}\}$, $sk_{\langle 0 \rangle} = (S_{\langle 0 \rangle}, \mathcal{Q}_{\langle 0 \rangle})$, and $\mathsf{Sec}_{\langle 0 \rangle} = (S_1, \ldots, S_{0^{\ell-1}1})$.)
7. Pick random $\mathsf{Sec}'_{\langle 0 \rangle}$ and $\mathsf{Sec}''_{\langle 0 \rangle}$ such that $\mathsf{Sec}_{\langle 0 \rangle} = \mathsf{Sec}'_{\langle 0 \rangle} + \mathsf{Sec}''_{\langle 0 \rangle}$ (i.e., for each $w \in \rho(\langle 0 \rangle)$ we have $S_w = S'_w + S''_w$). Set $SKB_{0.0} = \mathsf{Sec}'_{\langle 0 \rangle}$, $SK_{0.0} = (sk_{\langle 0 \rangle}, \mathsf{Sec}''_{\langle 0 \rangle})$.
8. Output $PK, SK_{0.0}$, and $SKB_{0.0}$ and erase all other information.

$\mathsf{UpdBase}(SKB_{t.r})$ does the following:

1. Parse $\langle t \rangle$ as $t_0 t_1 \cdots t_\ell$ where $t_0 = \varepsilon$ for convenience. Parse $SKB_{t.r}$ as $\mathsf{Sec}'_{\langle t \rangle} = \{S'_{t_0 \cdots t_{j-1} 1} \mid t_j = 0\}$.
2. If $t_\ell = 0$, erase $S'_{\langle t \rangle}$ and set $\mathsf{Sec}'_{\langle t+1 \rangle}$ equal to the remaining keys. Note that $S'_{\langle t+1 \rangle}$ is available, since it was stored as part of $\mathsf{Sec}_{\langle t \rangle}$.
3. Otherwise, let i be the largest value such that $t_i = 0$. Denote by $w = t_0 \cdots t_{i-1} 1$ the "deepest sibling" of t. For $j = 0, \ldots, \ell - i - 1$:
   (a) $s'_{w0^j} \leftarrow \mathbb{Z}_q$. Set $Q'_{w0^j} = s'_{w0^j} P$, $S'_{w0^j 0} = S'_{w0^j} + s'_{w0^j} H_1(w0^j 0)$, and $S'_{w0^j 1} = S'_{w0^j} + s'_{w0^j} H_1(w0^j 1)$.
4. Erase share S'_w and replace it by the resulting $(\ell - i)$ shares $\{S'_{w0^j 1}\}$ above, thus obtaining the new vector $SKB_{t+1.0} = \mathsf{Sec}'_{\langle t+1 \rangle}$.
5. Set $SKU_t = (S'_{\langle t+1 \rangle}, Q'_w, Q'_{w0}, \ldots, Q'_{w0^{\ell-i-1}})$. (Note that when $t_\ell = 0$ only $S'_{\langle t+1 \rangle}$ is sent.)
6. Output $SKB_{t+1.0}$, SKU_t.

$\mathsf{UpdUser}(SK_{t.r}, SKU_t)$ does the following:

1. Parse $\langle t\rangle$ as $t_0t_1\cdots t_\ell$ where $t_0 = \varepsilon$ for convenience. Parse $SK_{t.r}$ as $(S_{\langle t\rangle}, \mathcal{Q}_{\langle t\rangle}, \mathsf{Sec}''_{\langle t\rangle})$, where $\mathsf{Sec}''_{\langle t\rangle} = \{S''_{t_0\cdots t_{j-1}1} \mid t_j = 0\}$. Erase $S_{\langle t\rangle}$.
2. If $t_\ell = 0$, erase $S''_{\langle t\rangle}$ and set $\mathsf{Sec}''_{\langle t+1\rangle}$ equal to the remaining keys. Note that $S_{\langle t+1\rangle}$ is available, since it was stored as part of $\mathsf{Sec}_{\langle t\rangle}$.
3. Otherwise, let i be the largest value such that $t_i = 0$. Denote by $w = t_0\cdots t_{i-1}1$ the "deepest sibling" of t. Parse SKU_t as $(S'_{\langle t+1\rangle}, Q'_w, Q'_{w0}, \ldots, Q'_{w0^{\ell-i-1}})$. For $j = 0, \ldots, \ell-i-1$:
   (a) $s''_{w0^j} \leftarrow \mathbb{Z}_q$. Set $Q_{w0^j} = Q'_{w0^j} + s''_{w0^j}P$, $S''_{w0^j0} = S''_{w0^j} + s''_{w0^j}H_1(w0^j0)$, and $S''_{w0^j1} = S''_{w0^j} + s''_{w0^j}H_1(w0^j1)$.
4. Erase share S''_w and replace it by the resulting $(\ell-i)$ shares $\{S''_{w0^j1}\}$, thus obtaining the new vector $\mathsf{Sec}''_{\langle t+1\rangle}$. For $j = 0, \ldots, \ell-i-1$, erase $Q_{t_0\cdots t_{i-1}01^j}$ and replace it by $Q_{t_0\cdots t_{i-1}10^j}$. Thus, we obtain the the new vector $\mathcal{Q}_{\langle t+1\rangle}$.
5. Set $S_{\langle t+1\rangle} = S'_{\langle t+1\rangle} + S''_{\langle t+1\rangle}$.
6. Output $SK_{t+1.0} = (S_{\langle t+1\rangle}, \mathcal{Q}_{\langle t+1\rangle}, \mathsf{Sec}''_{\langle t+1\rangle})$.

$\mathsf{RefBase}(SKB_{t.r})$ does the following:

1. Parse $SKB_{t.r}$ as $\mathsf{Sec}'_{\langle t\rangle} = (\{S'_w \mid w \in \rho(t)\})$. For each $w \in \rho(t)$, pick random $R_w \in \mathbb{G}_1$ and reset each $S'_w := S'_w + R_w$.
2. Output resulting $SKB_{t.r+1}$ and $SKR_{t.r} = (\{R_w \mid w \in \rho(t)\})$.

$\mathsf{RefUser}(SK_{t.r}, SKR_{t.r})$ does the following:

1. Parse $SK_{t.r}$ as $(sk_{\langle t\rangle}, \mathsf{Sec}''_{\langle t\rangle})$, where $\mathsf{Sec}''_{\langle t\rangle} = (\{S''_w \mid w \in \rho(t)\})$. Parse $SKR_{t.r}$ as $(\{R_w \mid w \in \rho(t)\})$. For each $w \in \rho(t)$, reset $S''_w := S''_w - R_w$.
2. Output resulting $SK_{t.r+1}$.

$\mathsf{Enc}_{PK}(t, M)$ (where $M \in \{0,1\}^n$) does the following:

1. Let $t_1\cdots t_\ell = \langle t\rangle$. Select random $r \leftarrow \mathbb{Z}_q$.
2. Output $\langle t, C\rangle$ where:

$$C = (rP, rH_1(t_1t_2), \ldots, rH_1(t_1\cdots t_\ell), M \oplus H_2(\hat{e}(Q, H_1(t_1))^r))\,.$$

$\mathsf{Dec}_{SK_{t.r}}(\langle t, C\rangle)$ does the following:

1. Parse $\langle t\rangle$ as $t_1 \cdots t_\ell$. Parse $SK_{t.r}$ as $(sk_{\langle t\rangle}, \ldots)$ and $sk_{\langle t\rangle}$ as $(S_{\langle t\rangle}, \mathcal{Q}_{\langle t\rangle})$ where $\mathcal{Q}_{\langle t\rangle} = (Q_{t_1}, \ldots, Q_{t_1 \cdots t_{\ell-1}})$. Parse C as $(U_0, U_2, \ldots, U_\ell, V)$.
2. Compute

$$M = V \oplus H_2\left(\frac{\hat{e}(U_0, S_{\langle t\rangle})}{\prod_{j=2}^{\ell}\ \hat{e}(Q_{t_1\cdots t_{j-1}}, U_j)}\right).$$

We now verify that decryption is performed correctly. When encrypting, we have $\hat{e}(Q, H_1(t_1))^r = \hat{e}(P, H_1(t_1))^{rs_\varepsilon}$. When decrypting, we have $U_0 = rP$, $U_2 = rH_1(t_1t_2), \ldots, U_\ell = rH_1(t_1 \cdots t_\ell)$ so that

$$\begin{aligned}
\frac{\hat{e}(U_0, S_{\langle t\rangle})}{\prod_{j=2}^{\ell}\ \hat{e}(Q_{t_1\cdots t_{j-1}}, U_j)} &= \frac{\hat{e}\left(rP, s_\varepsilon H_1(t_1) + \sum_{j=2}^{\ell}\ s_{t_1\cdots t_{j-1}} H_1(t_1 \cdots t_j)\right)}{\prod_{j=2}^{\ell}\ \hat{e}\left(s_{t_1\cdots t_{j-1}} P, rH_1(t_1 \cdots t_j)\right)} \\
&= \frac{\hat{e}(P, H_1(t_1))^{rs_\varepsilon} \cdot \prod_{j=2}^{\ell}\ \hat{e}\left(P, H_1(t_1 \cdots t_j)\right)^{rs_{t_1\cdots t_{j-1}}}}{\prod_{j=2}^{\ell}\ \hat{e}\left(P, H_1(t_1 \cdots t_j)\right)^{rs_{t_1\cdots t_{j-1}}}} \\
&= \hat{e}(P, H_1(t_1))^{rs_\varepsilon}
\end{aligned}$$

and thus decryption succeeds.

3.3 Efficiency

Our scheme enjoys the same parameters as the forward-secure scheme of [8]. In fact, our scheme is exactly the "intrusion-resilient" extension of their forward-secure scheme. In particular, our scheme has public key of size $O(1)$, and all other parameters are $O(\log N)$ including: key generation time, encryption/decryption time, ciphertext length, key update time and message length, key refresh time and message length, and user/base storage.

3.4 Extensions

The full version of [8] gives a number of extensions of their original forward-secure scheme. In particular, they show (1) a modification of the scheme which can be proven secure in the standard model under the *decisional* BDH assumption; and (2) efficiency improvements which achieve key generation time and key update time $O(1)$. Both of these results may be carried over to our setting, via appropriate (small) modifications of the scheme presented above.

3.5 Security of Our Scheme

We now provide a sketch of the proof of security for the above scheme.

Theorem 1. *Under the computational BDH assumption (and in the random oracle model), the scheme described above is an intrusion-resilient public-key encryption scheme achieving semantic security.*

Proof. We convert any adversary A which successfully attacks the encryption scheme into an algorithm A' which breaks the BDH assumption. On a high level, A' will try to simulate the view of A in the following way: A' will guess the time period t^* for which A will ask its query to the LR oracle. If this guess turns out to be incorrect — in particular, as soon as A asks a query to LR which is *not* for time t^* or as soon as the scheme becomes (t^*, Q)-compromised — A' aborts the simulation and fails. On the other hand, if A' is correct in its guess (which occurs with probability $1/N$), then A' will be able to perfectly simulate the view of A in attacking the scheme. As in [8], this will enable A' to break the BDH assumption with probability $O(\mathsf{Adv}_A(k)/Nq_{H_2})$, where q_{H_2} is the number of hash queries A makes to H_2.

In more detail, adversary A' is given $(\mathbb{G}_1, \mathbb{G}_2, \hat{e})$ as output by $\mathcal{IG}(1^k)$, and is additionally given random elements $P, Q = s_\epsilon P, P' = bP$, and $U_0 = cP$. The goal of A' is to output $\hat{e}(P,P)^{s_\epsilon bc}$. A' will simulate an instance of the encryption scheme for adversary A. First, A' sets $PK = (\mathbb{G}_1, \mathbb{G}_2, \hat{e}, P, Q)$ and gives PK to A. Next, A' guesses a random index $t^* \in \{0, \ldots, N-1\}$ (this represents a guess of the period for which A will query LR). Let $\langle t^* \rangle = t_1^* \cdots t_\ell^*$ and $t_0^* = \epsilon$.

To answer the hash queries of A, algorithm A' maintains lists H_1^{list} and H_2^{list}. To begin, H_2^{list} will be empty. H_1^{list} is prepared by first having A' select random $x_2, \ldots, x_\ell \in \mathbb{Z}_q$ and then storing the tuples $(t_1^*,\ P')$, $(t_1^* t_2^*,\ x_2 P), \ldots,\ (t_1^* \cdots t_\ell^*,\ x_\ell P)$ in H_1^{list}. Next, A' proceeds as follows:

1. Choose random $y_1 \in \mathbb{Z}_q$ and store $(\overline{t_1^*}, y_1 P)$ in H_1^{list}.
2. For $2 \le k \le \ell$, choose random $y_k, s_k \in \mathbb{Z}_q$ and then store the value $\left(t_1^* \cdots t_{k-1}^* \overline{t_k^*}, y_k P - s_k^{-1} P'\right)$ in H_1^{list}.

A' will respond to hash queries of A in the obvious way. If A queries $H_b(X)$, then A' checks whether there is a tuple of the form (X, Y) in H_b^{list}. If so, the value Y is returned. Otherwise, A' chooses random Y from the appropriate range, stores (X, Y) in H_b^{list}, and returns Y.

We point out that fixing the output of the random oracle as above will allow A' to simulate the "local secret keys" for all nodes in the tree (as in [14, 8]) *except* those nodes on the path ρ from the root to leaf t^*.

In particular, fixing the output as above will allow A' to simulate local secret keys for all nodes whose parent is on path ρ; A' can then generate "real" local secret keys for the remaining nodes by following the legal description of the scheme.

Since all other values stored by the user and by the base are (individually) random, it should be clear that A' can simulate all queries of A to Osec *except* those for which the scheme becomes (t^*, Q)-compromised (where Q now represents the queries of A to Osec up to and including that point in time). When a query to Osec results in the scheme's becoming (t^*, Q)-compromised, A' simply aborts as its guess of t^* was incorrect; this will occur exactly with probability $(N-1)/N$. Assuming the scheme is never (t^*, Q)-compromised, A' can simulate the LR oracle as in [14, 8]. Specifically, A' will return the value $(U_0, x_2U_0, \ldots, x_\ell U_0, V)$, where V is a random element from $\{0,1\}^n$. Overall, with probability $1/N$, A' will be able to simulate the entire view of A. It is easy to see that the only way for A to get any advantage in the above simulation is to ask the random oracle H_2 the value $\hat{e}(P,P)^{s_\epsilon bc}$, as otherwise the encrypted message is information-theoretically hidden from A. Thus, outputting a random input element from the H_2^{list} will have non-negligible probability of being the value A' needs to break the BDH assumption.

3.6 Security Against Adaptive Chosen-Ciphertext Attacks

We briefly state our main results, and defer the details until the final version of our paper. To benefit from a modular approach (i.e., to avoid having to re-prove security every time a new scheme is constructed), we would like to have a generic transformation for securing intrusion-resilient public-key encryption schemes against adaptive chosen-ciphertext attacks. Such a transformation would take *any* intrusion-resilient public-key encryption scheme achieving semantic security and convert it to an intrusion-resilient public-key encryption scheme secure against adaptive chosen-ciphertext attacks. We call such a transformation a *CCA2-transformation.*

Much work along these lines has been done for the case of "standard" public-key encryption (e.g., [13, 22] and others), and many CCA2-transformations for "standard" public-key encryption schemes are known. The next theorem shows that we may leverage off this work for the case of intrusion-resilient encryption. Namely:

Theorem 2. *Any CCA2-transformation for "standard" public-key encryption schemes is also a CCA2-transformation for intrusion-resilient public-key encryption schemes.*

We note that this is different from the case of forward-secure public-key encryption, where such a result does not seem to hold [8].

Applying, e.g., the CCA2-transformation of [13, 22] to our scheme above, we obtain an intrusion-resilient public-key encryption scheme secure against adaptive chosen-ciphertext attacks.

References

1. M. Abdalla, S. Miner, and C. Namprempre. Forward-Secure Threshold Signature Schemes. RSA 2001.
2. M. Abdalla and L. Reyzin. A New Forward-Secure Digital Signature Scheme. Asiacrypt 2000.
3. R. Anderson. Two Remarks on Public-Key Cryptology. Invited lecture, CCCS '97. Available at `http://www.cl.cam.ac.uk/users/rja14/`.
4. M. Bellare, A Desai, E. Jokipii, and P. Rogaway. A Concrete Security Treatment of Symmetric Encryption. FOCS '97.
5. M. Bellare and S. Miner. A Forward-Secure Digital Signature Scheme. Crypto '99.
6. M. Bellare and A. Palacio. Protecting against Key Exposure: Strongly Key-Insulated Encryption with Optimal Threshold. Available at `http://eprint.iacr.org`.
7. D. Boneh and M. Franklin. Identity-Based Encryption from the Weil Pairing. Crypto 2001. Full version to appear in *SIAM J. Computing* and available at `http://eprint.iacr.org/2001/090/`.
8. R. Canetti, S. Halevi, and J. Katz. A Forward-Secure Public-Key Encryption Scheme. Preliminary version available at `http://eprint.iacr.org/2002/060/`.
9. A. De Santis, Y. Desmedt, Y. Frankel, and M. Yung. How to Share a Function Securely. STOC '94.
10. Y. Desmedt and Y. Frankel. Threshold Cryptosystems. Crypto '89.
11. Y. Dodis, J. Katz, S. Xu, and M. Yung. Key-Insulated Public-Key Cryptosystems. Eurocrypt 2002.
12. Y. Dodis, J. Katz, S. Xu, and M. Yung. Strong Key-Insulated Signature Schemes. PKC 2003.
13. E. Fujisaki and T. Okamoto. Secure Integration of Asymmetric and Symmetric Encryption Schemes. Crypto '99.
14. C. Gentry and A. Silverberg. Hierarchical ID-Based Cryptography. Asiacrypt 2002.
15. G. Itkis. Intrusion-Resilient Signatures: Generic Constructions, or Defeating a Strong Adversary with Minimal Assumptions. SCN 2002.
16. G. Itkis and L. Reyzin. Forward-Secure Signatures with Optimal Signing and Verifying. Crypto 2001.
17. G. Itkis and L. Reyzin. SiBIR: Signer-Base Intrusion-Resilient Signatures. Crypto 2002.
18. A. Joux. The Weil and Tate Pairing as Building Blocks for Public-Key Cryptosystems. ANTS 2002.
19. A. Joux and K. Nguyen. Separating Decision Diffie-Hellman from Diffie-Hellman in Cryptographic Groups. Manuscript, Jan. 2001. Available at `http://eprint.iacr.org`.
20. H. Krawczyk. Simple Forward-Secure Signatures From any Signature Scheme. CCCS 2000.

21. T. Malkin, D. Micciancio, and S. Miner. Efficient Generic Forward-Secure Signatures with an Unbounded Number of Time Periods. Eurocrypt 2002.
22. T. Okamoto and D. Pointcheval. REACT: Rapid Enhanced-Security Asymmetric Cryptosystem Transform. CT-RSA 2001.
23. R. Ostrovsky and M. Yung. How to Withstand Mobile Virus Attacks. PODC '91.