A Generic Construction for Intrusion-Resilient Public-Key Encryption

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

[1] Department of Computer Science, New York University. dodis@cs.nyu.edu
[2] University of California, Davis. franklin@cs.ucdavis.edu
[3] Department of Computer Science, University of Maryland. 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. In an *intrusion-resilient* cryptosystem [10], two entities (a user and a base) jointly evolve a secret decryption key; this provides very strong protection against an active attacker who can break into the user and base repeatedly and even simultaneously. Recently, a construction of an intrusion-resilient public-key encryption scheme based on specific algebraic assumptions has been shown [6]. We generalize this previous work and present a more generic construction for intrusion-resilient public-key encryption from any forward-secure public-key encryption scheme satisfying a certain homomorphic property.

1 Introduction

The exposure of secret keys can be a devastating attack against a cryptosystem. Especially when "standard" cryptanalytic techniques are infeasible, a determined attacker might find it much easier to obtain secret keys by hardware tampering, or via theft, bribery, or similar means. The problem of key exposure becomes more severe as cryptographic algorithms are increasingly used on inexpensive, lightweight, and portable consumer devices.

Key evolution is a powerful defense against the threat of key exposure. As an example, in a *forward-secure* scheme one's secret key is updated at each time period in such a way that key exposure during any time period compromises only future time periods (but not past time periods). Forward security was first formalized (in the context of signature and identification schemes) by Bellare and Miner [2], building on earlier ideas of Anderson [1]; numerous constructions of forward-secure signature schemes have been proposed (beginning with [2]). A forward-secure public-key encryption scheme has been constructed recently by Canetti, Halevi, and Katz [5].

Key-insulated cryptosystems [7, 3, 8] extend the key evolution paradigm to further limit the damage from key exposure. As with forward security, the user (e.g., a mobile device) can perform all cryptographic operations during any particular time period on his own. However, to update the user's secret keys for the next time period, the user needs the help of a "base" (e.g., a desktop PC in the user's home). Using this model, one may guarantee that exposure of the user's keys during multiple time periods only compromises security for those specific time periods, and not for any other time periods either in the past or in the future. A key-insulated scheme is additionally termed "strong" if there is no security compromise when the adversary exposes the secrets stored on the base.

Intrusion-resilience (first proposed in the context of signature schemes by Itkis and Reyzin [10]) is a synthesis of forward security and key-insulated security. The system model is as in the key-insulated case: the user performs cryptographic operations on its own during each time period, and updates its key for the next time period with the help of the base. Here, however, a stronger security guarantee is provided. If the base and the user are exposed during the *same* time period, then all *prior* time periods remain secure (as in the case of forward security). Otherwise, repeated exposure of both the user and the base only compromise those specific time periods during which the user's secret keys were exposed (as in the case of key-insulated security).

The security provided by intrusion-resilient schemes may be further enhanced by allowing "refresh" operations between base and user in addition to "update" operations. Both of these are key-evolving functions. The difference is that an update operation is used only at the beginning of each time period, while any number of refresh operations can occur within a single time period. Someone who wants to interact with the user needs to know the current time period (i.e., number of update operations), but does not need to know how many refresh operations have occurred within each time period. Frequent refresh operations enhance security, since the attacker must expose user and base *between refreshes* in order to compromise future security.

Itkis and Reyzin [10] gave a construction of intrusion-resilient signatures based on the strong RSA assumption. Subsequently, Itkis [9] showed a *generic* construction of intrusion-resilient signatures from any one-way function. The first construction for intrusion-resilient public-key encryption is given in [6]. That construction relies on a very specific assumption (the BDH assumption [4]), and is based on the forward-secure encryption scheme of [5]. This raises the natural question of what assumptions are sufficient to achieve intrusion-resilient encryption. In this paper, we make progress on this question by presenting a more generic construction for intrusion-resilient public key encryption based on any forward-secure encryption scheme satisfying certain properties. In this sense, our work generalizes the previous work [6] which constructs an intrusion-resilient encryption schemes from a *specific* forward-secure scheme (i.e., that of [5]). It is hoped that our more generic construction will highlight those properties that enable intrusion-resilience and thus shed additional light on this primitive.

Indeed, the scheme in [6] is somewhat complicated and hard to parse. In particular, one has to be extremely careful when defining the order of operations in that scheme, it is not immediately really clear what specific properties of the forward-secure scheme of [5] are critically used, and, overall, what is the high level intuition behind that construction. This paper tries to clarify this point by presenting a more generic construction of intrusion-resilient encryption which clearly explains which special properties of the scheme of [5] are used. Specifically, we isolate two such crucial properties: a homomorphic structure of the key updating operation, and, more importantly, "separability" between the user's key material used for updating from that used for the actual decryption. Indeed, we will argue that without such separability it seems impossible (or very hard) to build an intrusion-resilient encryption scheme from a forward-secure scheme. For that reason, we also give a new, refined definition of forward-secure encryption which explicitly models this key separability, and argue that the scheme in [5] meets our definition. Then, we give a clean and intuitive construction of intrusion-resilient encryption from any such refined forward-secure encryption with an extra homomorphic property for key updating. Of course, since presently there exists only one specific forward-secure encryption of [5], we can currently instantiate our scheme in only one way — the one given in [6] — but our exposition hopefully clarifies and explains the design criteria for constructing intrusion-resilient encryption. In particular, shall a new forward-secure scheme be found, our construction pin-points the two natural extra properties which are needed to turn it into an intrusion-resilient scheme (from the same assumption). And since we argue that such extra properties also seem to be necessary, our work motivates the design of future forward-secure schemes which satisfy them as well. Thus, we believe that our generally will clarify and simplify future designs of both forward-secure and intrusion-resilient schemes.

As an additional contribution, we explore a number of alternative models and definitions for both forward-secure and intrusion-resilient encryption in Sections 2 and 3. Our generic construction appears in Section 4, and Section 5 contains the proof of security.

2 Forward-Secure Encryption

Our intrusion-resilient scheme is built from a forward-secure encryption scheme. The notation and model borrows from that of [5], slightly adapted for our purposes. We let $\mathbb{N}$ be the set of positive integers and let $[T] = \{1, 2, \ldots, T\}$.

2.1 Functional Description

We assume a key-evolving encryption scheme in which the user's secret key can be "divided" into two components: an *update key* and a *local key*. An update key is used only to generate the update key and local key of the next time period, but is not used to decrypt a ciphertext. On the other hand, a local key is used only to decrypt a ciphertext in the corresponding time period, but is not used to generate the update key or local key for the next time period. Note that the forward-secure encryption scheme of [5] may be viewed in this way.

More formally, we specify a key-evolving encryption scheme (with the above-mentioned property) by the following tuple of polynomial-time algorithms:

`fsKeyGen`: key generation algorithm
Input: security parameter k, number of time periods T
Output: initial user key $\overline{sk}_0$, public key $\overline{pk}$

`fsKeyUpd`: key-update algorithm
Input: current user update key $\overline{sk}_t$ and time period t
Output: next user update key $\overline{sk}_{t+1}$ and next user local key $\overline{lsk}_{t+1}$

`fsEnc`: randomized encryption algorithm
Input: user public key $\overline{pk}$, current time period t, message M
Output: ciphertext C

`fsDec`: decryption algorithm
Input: user local secret key $\overline{lsk}_t$, ciphertext $C = \mathtt{fsEnc}(\overline{pk}, t, M)$
Output: message M

The initial user update key $\overline{sk}_0$ is not actually used or stored (instead, `fsKeyUpd` is applied immediately to generate $\overline{sk}_1$ and $\overline{lsk}_1$). Therefore, the sets of keys which an adversary can access are defined as follows:

$$\overline{sk}^* = \{\overline{sk}_t | 1 \leq t \leq T\} \text{ and } \overline{lsk}^* = \{\overline{lsk}_t | 1 \leq t \leq T\}.$$

Remark: Note that in the definition of [5], a single secret key is used both for updates and for decryption (instead of having separate keys for updates and decryption, as above). We call such a scheme a *primitive* key-evolving scheme. A primitive scheme which is forward-secure is called a `PFSE` scheme, to distinguish it from forward-secure schemes which can additionally be cast as per the above definition (these are called `FSE` schemes).

2.2 Definition of Security

We now provide a definition of forward security for a key-evolving encryption scheme as defined in the previous section. Our definition is stronger than than the definition given in [5] in that we allow the adversary to obtain the local key (but not the update key) for time periods prior to the challenge time period. Formally, we accomplish this by giving the adversary access to two separate oracles: one of which returns local keys, and one of which returns update keys. Although this is a stronger definition than that given previously, note that the scheme of [5] satisfies it.

Let A be a probabilistic polynomial-time oracle Turing machine, which gets input $\overline{pk}$ and T, and interacts with the following oracles:

• Decryption oracle $O_{FSDec}(\overline{sk}_0, \cdot, \cdot)$, which on input $t \in [T]$ and a ciphertext C outputs a message M decrypted by $\overline{lsk}_t$ (where this key is derived in the appropriate way from $\overline{sk}_0$).

• Update-key oracle $O_{FSukey}(\overline{sk}_0, \cdot)$, which on input $t \in [T]$ outputs $\overline{sk}_t$ (again, this key is derived in the appropriate way from $\overline{sk}_0$).

• Local-key oracle $O_{FSlkey}(\overline{sk}_0, \cdot)$, which on input $t \in [T]$ outputs $\overline{lsk}_t$ (again, this key is derived in the appropriate way from $\overline{sk}_0$).

• Left-or-right oracle $O_{FSLR}(\overline{pk}, \cdot, LR_b(\cdot, \cdot))$ which on inputs $t^* \in [T]$ and equal-length messages m_0, m_1 returns a challenge ciphertext $C^* \leftarrow \mathtt{fsEnc}(\overline{pk}, t^*, m_b)$. The bit b is chosen randomly at the outset of the experiment.

The adversary A may query all oracles adaptively, in any order it wants, subject to the following restrictions: queries t to O_{FSukey} satisfy $t > t^*$; queries t' to O_{FSlkey} satisfy $t' \neq t^*$; only a single query is made to O_{FSLR}; and the ciphertext C^* received from O_{FSLR} may not be queried to O_{FSDec} for time period t^*. Eventually, the adversary guesses a bit b' and halts. The adversary succeeds if $b' = b$. We define the adversary's advantage as the absolute value of the difference between its success probability and 1/2.

Definition 1. *We say that a key-evolving encryption scheme* `FSE` *is forward secure against chosen-ciphertext attacks (*`FS-CCA`*) if the advantage of any* PPT *adversary A in the above experiment is negligible.*

Remark: We stress that separating the two oracles O_{FSukey} and O_{FSlkey} strengthens the notion of forward security as compared to [5]. Specifically, our model allows an adversary to get the local key corresponding to any $t' \neq t^*$.

3 Intrusion-Resilient Encryption

As mentioned in the introduction, intrusion-resilient encryption schemes achieve a stronger level of security than forward-secure encryption schemes, at the cost of introducing a second entity (i.e., the base). Our definition of security follows [10, 6]. An adversary is allowed an adaptive chosen-ciphertext attack, can additionally obtain the secrets from the base and/or the user, and can eavesdrop on the communication between the base and user. As long as the user, the base, and the communication between user and base are not compromised at the same time period, the scheme remains secure for all time periods at which the user's key was not exposed. Furthermore, the scheme achieves forward security in case the user, base, and communication between user and base *are* compromised at the same time period. We now provide formal definitions.

3.1 Functional Description

The encryption scheme is specified by the following tuple of polynomial-time algorithms:

`KeyGen`: key generation algorithm
Input: security parameter k, number of time periods T, number of refreshes R
Output: initial user key $sk_{0.0}$, initial base key $skb_{0.0}$, public key pk

`BaseUpd`: base key-update algorithm
Input: current base key $skb_{t.r}$
Output: next base key $skb_{t+1.0}$, key update message sku_t

`UserUpd`: user key-update algorithm
Input: current user key $sk_{t.r}$, key update message sku_t
Output: next user key $sk_{t+1.0}$

`BaseRef`: base key-refresh algorithm
Input: current base key $skb_{t.r}$
Output: next base key $skb_{t.r+1}$, corresponding key refresh message $skr_{t.r}$

`UserRef`: user key-refresh algorithm
Input: current user key $sk_{t.r}$, key refresh message $skr_{t.r}$
Output: next user key $sk_{t.r+1}$

`Enc`: randomized encryption algorithm
Input: user public key pk, current time interval t, message M
Output: ciphertext C

`Dec`: decryption algorithm
Input: user secret key $sk_{t.r}$, ciphertext $C = \texttt{Enc}(pk, t, M)$
Output: message M

The encryption scheme is run as follows:

```
Syntactic(k, T, R)
  Set (sk_{0.0}, skb_{0.0}, pk) ← KeyGen(k, T, R).
  For t = 0 to T − 1:
   Set (skb_{t+1.0}, sku_t) ← BaseUpd(skb_{t.r}) and sk_{t+1.0} ← UserUpd(sk_{t.r}, sku_t).
   For r = 0 to R − 1
    Set (skb_{t.r+1}, skr_{t.r}) ← BaseRef(skb_{t.r}) and sk_{t.r+1} ← UserRef(sk_{t.r}, skr_{t.r}).
```

Here the keys $sk_{t,0}$ and $skb_{t,0}$ for $0 \leq t \leq T$ are not actually used or stored. Key generation is immediately followed by an update, and each update is immediately followed by a refresh. Therefore, the secret keys which an adversary can potentially access are defined as follows:

- $sk^* = \{sk_{t.r} | 1 \leq t \leq T, 1 \leq r \leq R\}$
- $skb^* = \{skb_{t.r} | 1 \leq t \leq T, 1 \leq r \leq R\}$
- $sku^* = \{sku_t | 1 \leq t \leq T - 1\}$
- $skr^* = \{skr_{t.r} | 1 \leq t \leq T - 1, 0 \leq r \leq R - 1\} \setminus \{skr_{1.0}\}$

3.2 Definition of Security

We now define intrusion-resilience. Let A be a probabilistic polynomial-time oracle Turing machine which gets input pk, T, and R, and which may query the following oracles (each oracle is technically indexed by an initial tuple of keys $(sk_{0.0}, skb_{0.0}, pk)$ which is omitted for readability):

- Decryption oracle O_{Dec}, which on input $t \in [T]$, $r \in [R]$, and a ciphertext C outputs a message M decrypted using $sk_{t.r}$
- User key oracle O_{sk}, which on input $t \in [T]$ and $r \in [R]$ outputs $sk_{t.r}$
- Base key oracle O_{bk}, which on input $t \in [T]$ and $r \in [R]$ outputs $skb_{t.r}$
- Key update oracle O_u, which on input $t \in [T]$ outputs sku_t
- Key refresh oracle O_r, which on input $t \in [T]$ and $r \in [R]$ outputs $skr_{t.r}$

• Left-or-right oracle O_{LR}, which on input $t^* \in [T]$ and equal-length messages m_0, m_1, outputs challenge ciphertext $C^* \leftarrow \texttt{Enc}(pk, t^*, m_b)$ (for a bit b which is chosen at random at the beginning of the experiment).

The oracles O_{sk}, O_{bk}, O_u and O_r are generically called "key exposure oracles", and are denoted by O_{sec}. Queries to a particular oracle are indicated by including the appropriate string; thus, $O_{sec}(\text{"sk"}, t.r)$ denotes the query $O_{sk}(t, r)$.

The only restrictions for the adversary's queries are that key exposures must *respect erasure*. That is, if a value corresponding to a particular instant in time t_1 has been obtained by the adversary (via an oracle query), then a value corresponding to a prior instant in time (which would have been erased prior to t_1) cannot be obtained. More formally,

◇ ("sk", $t.r$) must be queried before ("sk", $t'.r'$) if $t' > t$ or $t' = t$ and $r' > r$;
◇ ("bk", $t.r$) must be queried before ("bk", $t'.r'$) if $t' > t$ or $t' = t$ and $r' > r$;
◇ ("bk", $t.r$) must be queried before ("r", $t'.r'$) if $t' > t$ or $t' = t$ and $r' \geq r$;
◇ ("bk", $t.r$) must be queried before ("u", t') if $t' \geq t$.

For a set Q of key exposure queries, we say that $sk_{t.r}$ is Q-*exposed* if on of the following is true:

– ("sk", $t.r$) $\in Q$;
– $r > 1$, ("r", $t.(r-1)$) $\in Q$, and $sk_{t.(r-1)}$ is Q-exposed;
– $r = 1$, ("u", $t-1$) $\in Q$, and $sk_{(t-1).R}$ is Q-exposed;
– $r < R$, ("r", $t.r$) $\in Q$, and $sk_{t.r+1}$ is Q-exposed.

A completely analogous definition may be given for Q-exposure of a base key $skb_{t.r}$. We say the scheme is (t^*, Q)-*compromised* if $sk_{t^*.r}$ 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 that an adversary *succeeds* if it correctly guesses the bit b used by the O_{LR} oracle, subject to the following restrictions: (1) The system was not (t^*, Q)-compromised where O_{LR} was queried at time period t^*; and (2) The ciphertext C^* returned by O_{LR} was not queried to O_{Dec} (for the same time period t^*). An adversary's advantage is defined as the absolute value of the difference between its success probability and 1/2.

Definition 2. *We say that an encryption scheme is intrusion-resilient against chosen-ciphertext attacks (*`IR-CCA`*) if the advantage of any* PPT *adversary A in the above experiment in negligible.*

Remark: We sometimes refer to the notion defined above as "full" intrusion resilience. In Appendix A, we define a security notion called *quasi-intrusion resilience* which lies "in between" key-insulated security and full intrusion-resilience. This intermediate notion helps describe the security level which is achieved by using a *primitive* key-evolving encryption scheme.

4 A Generic Construction of Intrusion-Resilient Encryption

In this section, we present a generic construction of a fully intrusion-resilient encryption scheme from

4.1 Preparations

a forward secure encryption scheme whose key-update algorithm is homomorphic in the sense we now describe. Assume a map

$$\phi : G_1 \to G_2 \times G_3,$$

where G_1, G_2, and G_3 are groups represented additively. We say that the map ϕ is homomorphic if for all $x, y \in G_1$ we have:

$$\phi(x + y) = \phi(x) + \phi(y).$$

More precisely, ϕ satisfies

$$\phi(x + y) = (x_1 + y_1, x_2 + y_2)$$

where $\phi(x) = (x_1, x_2)$ and $\phi(y) = (y_1, y_2)$.

To give a generic construction of fully intrusion resilient scheme, we specify the key-evolving encryption scheme FSE as generally as possible. Let S_1 be a set, and let G_1, G_2, and G_3 be groups (written additively). Let FSE be as follows:

FSE $=$ (fsKeyGen, fsKeyUpd, fsEnc, fsDec):

- fsKeyGen: $\{0,1\}^* \times \mathbb{N} \to G_1 \times S_1$; fsKeyGen$(k, T)$=$(\overline{sk}_0, \overline{pk})$
- fsKeyUpd : $G_1 \to G_1 \times G_2$; fsKeyUpd$(\overline{sk}_t) = (\overline{sk}_{t+1}, \overline{lsk}_{t+1})$

Additionally, fsKeyUpd should be homomorphic; that is:

$$\texttt{fsKeyUpd}(x + y) = \texttt{fsKeyUpd}(x) + \texttt{fsKeyUpd}(y).$$

In other words, it satisfies:

$$\texttt{fsKeyUpd}(x + y) = (x_1 + y_1, x_2 + y_2),$$

where fsKeyUpd$(x) = (x_1, x_2)$ and fsKeyUpd$(y) = (y_1, y_2)$.

- fsEnc : $S_1 \times \mathbb{N} \times \{0,1\}^n \to \{0,1\}^n$; fsEnc$(\overline{pk}, t, M) = C$
- fsDec : $G_2 \times \{0,1\}^n \to \{0,1\}^n$; fsDec$(\overline{lsk}_t, C) = M$

4.2 Scheme Intuition

As intuition for our construction, we may note that a secret key of encryption scheme FSE consists of $\overline{sk}_t$ and $\overline{lsk}_t$, where the local key $\overline{lsk}_t$ is used only for decryption. We may notice that a user update key $\overline{sk}_t$ of FSE enables derivation of all the user secret keys for periods t through N, but none of the secret keys for periods $t' < t$. This will allow us to achieve forward security, as in [5]. However, in our model we also need to divide the user update key between the user and the base, so that we can derive the sharing for period $t + 1$ from that of period t and achieve future security also. To achieve this, we let the user store $\overline{lsk}_t$ — to enable decryption within the current time period — but additively share the user update key $\overline{sk}_t$ between the user and the base. In summary, let the user store $\overline{lsk_t}$ and the evolved share of $\overline{sk_t}$, and the base store the other evolved share of $\overline{sk_t}$. Intuitively, $\overline{lsk_t}$ by itself only allows the user to decrypt at period t, and the fact that the user update key $\overline{sk_t}$ is split ensures that exposure of the user cannot compromise any of the future periods. Security against compromises of the base follows similarly. This gives us intrusion-resilience.

The only issue to resolve is how to update a local key by using the separated shares of an update key. Both shares of the user and the base are evolved in each time period, which are executed independently by the user and the base. When each share is evolved by using the key-update algorithm of `FSE`, the algorithm outputs two elements: the sharing of the next-time-period update key and the sharing of the next-time-period local key. The base sends only the sharing of a local key to the user as the update message, and the user combines it with his own sharing of the local key by using the homomorphic property of the key-update algorithm; thus, the user derives the the next-time-period local key. As a result, the user and the base generate their own update keys independently and compute the next-time-period local key jointly. This step is immediately followed by a random refresh.

4.3 FISER

We now describe the fully intrusion-resilient encryption scheme `FISER` = (`KeyGen`, `BaseUpd`, `UserUpd`, `BaseRef`, `UserRef`, `Enc`, `Dec`). Let us note that each parameter is defined on the following set or groups:

◇ set of user public keys :S_1
◇ group of user secret keys :$G_3 = G_1 \times G_2$
◇ group of base secret keys :G_1
◇ group of key update message :$G_3 = G_1 \times G_2$
◇ group of key refresh message :G_1

Using the above notation, each function is described as follows:

`KeyGen`: $\{0,1\}^* \times \mathbb{N} \to G_3 \times G_1 \times S_1$; `KeyGen`$(k, T)$=$(sk_{0.0}, skb_{0.0}, pk)$
1. Compute $(\overline{sk}_0, \overline{pk}) \leftarrow$ `fsKeyGen`(k, T).
2. Let $\overline{sk}_0$ be divided in $\overline{sk}_0 = \overline{sks}_{0.0} + \overline{skb}_{0.0}$ for randomly chosen $\overline{sks}_{0.0} \in G_1$
3. Set $pk = \overline{pk}$, $sk_{0.0} = (\overline{sks}_{0.0}, \cdot)$, and $skb_{0.0} = \overline{skb}_{0.0}$,
4. Output $sk_{0.0}$, $skb_{0.0}$, and pk.

`BaseUpd`: $G_1 \to G_1 \times G_2$; `BaseUpd`$(skb_{t.r}) = (skb_{t+1.0}, sku_t)$
For an input of base secret key $skb_{t.r} = \overline{skb}_{t.r}$
1. Compute $(\overline{skb}_{t+1.0}, sku_t) \leftarrow$ `fsKeyUpd`$(\overline{skb}_{t.r})$.
2. Output $skb_{t+1.0} = \overline{skb}_{t+1.0}$ and sku_t.

`UserUpd`: $G_3 \times G_2 \to G_3$; `UserUpd`$(sk_{t.r}, sku_t) = sk_{t+1.0}$
For inputs of user secret key $sk_{t.r} = (\overline{sks}_{t.r}, \overline{lsk}_t)$ and update message sku_t
1. Compute $(\overline{sks}_{t+1.0}, lsk_{t+1}) \leftarrow$ `fsKeyUpd`$(\overline{sks}_{t.r})$.
2. Compute $\overline{lsk}_{t+1} = lsk_{t+1} + sku_t$.
3. Output $sk_{t+1.0} = (\overline{sks}_{t+1.0}, \overline{lsk}_{t+1})$.

`BaseRef`: $G_1 \to G_1 \times G_1$; `BaseRef`$(skb_{t.r}) = (skb_{t.r+1}, skr_{t.r})$
For an input of base secret key $skb_{t.r} = \overline{skb}_{t.r}$,

1. Compute $\overline{skb}_{t.r+1} = \overline{skb}_{t.r} - \overline{R}_{t.r}$ for a random secret $\overline{R}_{t.r} \in G_1$.
2. Output $skb_{t.r+1} = \overline{skb}_{t.r+1}$ and $skr_{t.r} = \overline{R}_{t.r}$.

`UserRef`: $G_3 \times G_1 \to G_3$; $\mathtt{UserRef}(sk_{t.r}, skr_{t.r}) = sk_{t.r+1}$

For inputs of user secret key $sk_{t.r} = (\overline{sks}_{t.r}, \overline{lsk}_t)$ and refresh message $skr_{t.r} = \overline{R}_{t.r}$,
1. Compute $\overline{sks}_{t.r+1} = \overline{sks}_{t.r} + \overline{R}_{t.r}$.
2. Output $sk_{t.r+1} = (\overline{sks}_{t.r+1}, \overline{lsk}_t)$.

`Enc`: $S_2 \times \mathbb{N} \times \{0,1\}^n \to \{0,1\}^n$; $\mathtt{Enc}(pk, t, M) = C$

For inputs of a public key pk, time t, and a message M,
1. Compute $C \leftarrow \mathtt{fsEnc}(pk, t, M)$.
2. Output C.

`Dec`: $G_3 \times \{0,1\}^n \to \{0,1\}^n$; $\mathtt{Dec}(sks_t, C) = M$

For inputs of user secret key $sk_{t.r} = (\overline{sks}_{t.r}, \overline{lsk}_t)$ and ciphertext C,
1. Compute $M \leftarrow \mathtt{fsDec}(\overline{lks}_t, C)$.
2. Output M.

5 Security Analysis

We now prove security of the `FISER` given above. For simplicity, the time complexity of an adversary A is defined as the execution time of the experiment used to define the advantage of A, including the time taken for key generation and initialization, as well as the time required for the various oracles to compute replies to the adversary's queries.

Theorem 1. *Let A be an adversary of time complexity τ with at most Q queries to oracles $O \in \{O_{Dec}, O_{sec}, O_{LR}\}$ against* `FISER`*. If A has advantage δ, then there exists an adversary B performing a chosen-ciphertext attack against the underlying* `FSE` *with at least the same advantage. The time complexity of B is at most $\tau + O(\log k)$, and the number of queries is at most Q.*

Proof. We construct an adversary B that uses A to perform a chosen-ciphertext-and-key attack against `FSE`. B is allowed to ask queries to: a decryption oracle $O_{FSDec}(\cdot, \overline{pk}, \overline{sk}_{t.r}, \cdot)$; a user update-key oracle $O_{FSukey}(\overline{pk}, \overline{sk}_0, \cdot)$; a user local-key oracle $O_{FSlkey}(\overline{pk}, \overline{sk}_0, \cdot)$; and a left-or-right oracle $O_{FSLR}(\overline{pk}, \cdot, LR(\cdot, \cdot, b))$. Adversary B receives challenge ciphertext $C^* = \mathtt{fsEnc}(\overline{pk}, t^*, m_b)$, and outputs a guess b'. Adversary B succeeds if $b' = b$.

B simulates A's environment as follows: first, B runs A until A outputs T and $R \in \mathbb{N}$. B also returns T. B runs $\mathtt{fsKeyGen}(k, T)$ to produce $(\overline{sk}_0, \overline{pk})$. B chooses $\overline{skb}_{0.0} \in G_1$ randomly and maintains a list U_1^{list}, which consists of tuples of the following form:

$$(t, r; \overline{sks}_{t.r}, \overline{skb}_{t.r}, \overline{R}_{t.r}) \in \mathbb{N} \times \mathbb{N} \times G_1 \times G_1 \times G_1.$$

We use the notation $(t, r; \overline{sks}_{t.r}, -, *)$ as follows: "$-$" is used if there is no list on $\overline{skb}_{t.r}$, i.e. empty , and "*" is used if we don't care about $\overline{R}_{t.r}$ like empty or not, or if we maintain the data after some operating. For example, "change $(t, r; *, *, -)$ to $(t, r; *, *, \overline{R}_{t.r})$" means that: change the data "-" to $\overline{R}_{t.r}$ while maintaining the data of $\overline{sks}_{t.r}$ and $\overline{skb}_{t.r}$ as they are.

To begin, B sets $pk = \overline{pk}$ and $U_1^{list} = \{(0, 0; -, \overline{skb}_{0.0}, -)\}$ and continues the execution of A on input pk using its oracles to respond A's queries as follows:

Decryption oracle. Let a query to $O_{Dec}(\cdot, \cdot, pk, sk_{t.r}, \cdot)$ be (t, r, C). B forwards (t, C) to its decryption oracle $O_{FSDec}(\cdot, \overline{pk}, \overline{sk}_0, \cdot)$, and returns the answer M to A. From the definition of O_{Dec}, the answer is exactly what A's decryption oracle would have answered.

Base key oracle. Let a query to $O_{bk}(\overline{skb}_{0.0}, pk, \cdot, \cdot)$ be (t, r). B conducts the following steps.

1. If there is $(t, r; *, \overline{skb}_{t.r}, *)$ in U_1^{list}, then pick $\overline{skb}_{t.r}$ from U_1^{list}.
2. Else if there is $(t, r; \overline{sks}_{t.r}, -, *)$ in U_1^{list}, which means exactly "simultaneous attack", then forward t to its user update-key oracle O_{FSukey}, get the answer $\overline{sk}_t$, compute
$$\overline{skb}_{t.r} = \overline{sk}_t - \overline{sks}_{t.r},$$
and renew U_1^{list} by using $(t, r; \overline{sks}_{t.r}, \overline{skb}_{t.r}, *)$ instead of $(t, r; \overline{sks}_{t.r}, -, *)$.
3. Else if $r > 1$ and there is $(t, r-1; *, \overline{skb}_{t.r-1}, \overline{R}_{t.r-1})$ in U_1^{list}, then compute
$$\overline{skb}_{t.r} = \overline{skb}_{t.r-1} - \overline{R}_{t.r-1}$$
and renew U_1^{list} by using $(t, r; *, \overline{skb}_{t.r}, *)$.
4. Otherwise, choose $\overline{skb}_{t.r} \in G_1$ randomly and renew U_1^{list} using $(t, r; -, \overline{skb}_{t.r}, *)$ in U_1^{list}.
5. Finally B returns $\overline{skb}_{t.r}$ to A.

Since $\overline{skb}_{t.r}$ was exactly what A's base key oracle would have answered, A's view is identical to its view in the attack against `FISER`.

User key oracle. Let a query to $O_{sk}(sk_{0.0}, pk, \cdot, \cdot)$ be (t, r). B conducts the following steps.

1. If there is $(t, r; \overline{sks}_{t.r}, *, *)$ in U_1^{list}, then pick $\overline{sks}_{t.r}$ from U_1^{list}.
2. Else if there is $(t, r; -, \overline{skb}_{t.r}, *)$ in U_1^{list}, which means exactly "simultaneous attack", then forward t to its user update key oracle O_{FSukey}, get the answer $\overline{sk}_t$, compute
$$\overline{sks}_{t.r} = \overline{sk}_t - \overline{skb}_{t.r},$$
and renew U_1^{list} by using $(t, r; \overline{sks}_{t.r}, \overline{skb}_{t.r}, *)$.
3. Else if $r > 1$ and there is $(t, r; \overline{sks}_{t.r-1}, *, \overline{R}_{t.r-1})$ in U_1^{list}, then compute
$$\overline{sks}_{t.r} = \overline{sks}_{t.r-1} + \overline{R}_{t.r-1}$$
and renew U_1^{list} by using $(t, r; \overline{sks}_{t.r}, -, *)$.
4. Otherwise, choose $\overline{sks}_{t.r} \in G_1$ randomly and renew U_1^{list} using $(t, r; \overline{sks}_{t.r}, -, *)$.
5. Finally B returns $\overline{sks}_{t.r}$ to A.

Since $\overline{sks}_{t.r}$ was exactly what A's user key oracle would have answered, A's view is identical to its view in the attack against FISER.

Refresh oracle. Let a query to $O_r(skb_{0.0}, pk, \cdot, \cdot)$ be (t, r). B conducts the following steps.

1. If there is $(t, r; *, *, \overline{R}_{t.r})$ in U_1^{list}, then pick $\overline{R}_{t.r}$ from U_1^{list}.
2. Else if either of the following are in U_1^{list}:
$$\{(t, r; \overline{sks}_{t.r}, *, -), (t, r; \overline{sks}_{t.r+1}, *, -)\}$$
or
$$\{(t, r; *, \overline{skb}_{t.r}, *), (t, r; *, \overline{skb}_{t.r+1}, *)\},$$
then compute
$$\overline{R}_{t.r} = \overline{sks}_{t.r+1} - \overline{sks}_{t.r} \text{ or } \overline{R}_{t.r} = \overline{skb}_{t.r} - \overline{skb}_{t.r+1},$$
and renew U_1^{list} by using $(t, r; \overline{sks}_{t.r}, *, \overline{R}_{t.r})$ or $(t, r; *, \overline{skb}_{t.r}, \overline{R}_{t.r})$, respectively.
3. Otherwise, choose $\overline{R}_{t.r} \in G_1$ randomly and renew U_1^{list} using $(t, r; *, *, \overline{R}_{t.r})$.
4. Finally B returns $\overline{R}_{t.r}$ to A.

Since $\overline{R}_{t.r}$ was exactly what A's refresh oracle would have answered, A's view is identical to its view in the attack against FISER.

Update oracle. Let a query to $O_u(skb_{0,0}, pk, \cdot)$ be t. B does as follows:

1. If there is $(t, R; *, \overline{skb}_{t.R}, *)$ in U_1^{list}, then compute
$$(\overline{skb}_{t+1.0}, sku_t) \leftarrow \texttt{fsKeyUpd}(\overline{skb}_{t.R}).$$
2. Else if there is $(t, R; \overline{sks}_{t.R}, -, *)$ in U_1^{list}, forward $t+1$ to its local-key oracle $O_{FSlkey}(\overline{pk}, \overline{sk}_0, \cdot)$, obtain the answer $\overline{lsk}_{t+1}$, and compute
$$\begin{aligned}(\overline{sks}_{t+1.0}, lsk_{t+1}) &\leftarrow \texttt{fsKeyUpd}(\overline{sks}_{t.R}) \text{ and}\\ sku_t &= \overline{lsk}_{t+1} - lsk_{t+1}.\end{aligned}$$
3. Otherwise, choose randomly $\overline{sks}_{t.R} \in G_1$, forward $t+1$ to its local-key oracle $O_{FSlkey}(\overline{pk}, \overline{sk}_0, \cdot)$, obtain the answer $\overline{lsk}_{t+1}$, compute
$$\begin{aligned}(\overline{sks}_{t+1.0}, lsk_{t+1}) &\leftarrow \texttt{fsKeyUpd}(\overline{sks}_{t.R}) \text{ and}\\ sku_t &= \overline{lsk}_{t+1} - lsk_{t+1},\end{aligned}$$
and renew U_1^{list} by using $(t, r; \overline{sks}_{t.R}, -, *)$.
4. Finally B returns sku_t to A.

Since sku_t was exactly what A's update key oracle would have answered, A's view is identical to its view in the attack against FISER.

Left-or-right oracle. Let a query to $O_{LR}(pk, \cdot, LR(\cdot, \cdot, b))$ be (t^*, m_0, m_1). Then B forwards (t^*, m_0, m_1) to its left-or-right oracle $O_{FSLR}(\overline{pk}, \cdot, LR(\cdot, \cdot, b))$, obtains a ciphertext C^*, and returns C^* to A. From the definition of Enc, the answer is exactly what A's left-or-right oracle would have answered.

When A outputs its guess bit b' and halts, B returns b' and halts. Note that even if A makes queries to more than one oracle $O \in O_{sec}$ for the same time/refresh period (t, r), adversary A does not see any inconsistencies among the answers from these oracle queries unless the scheme becomes (t^*, Q)-compromised (where Q represents the queries of A to O_{sec} up to and including that point in time); this assumes that A is respects erasure. Furthermore, B queries O_{FSukey} if and only if A queries both O_{sk} and O_{bk} for the same time/refresh period (t, r). That is, the earliest time period queried to both O_{sk} and O_{bk} simultaneously by A is coincident with that time period submitted to O_{FSukey} by B. Therefore B succeeds whenever A does.

From the above simulation by B, we see that the time complexity of B is at most $\tau + \log k$ and that B makes at most Q queries to its oracles.

6 Further Discussion

There are several security notions of key-evolving or key-updating encryption schemes: forward-secure encryption as defined by [5] (called PFSE), forward-secure encryption (FSE) as defined here (recall, in our model the secret key is split into a key used for decryption and a key used for updates), key-insulated encryption [7], and intrusion-resilient encryption [6]. These notions and the notion of ID-based encryption (IBE) [4] are related; this has already been noted in [7, 3, 8]. We summarize the relation here.

Any secure ID-based encryption scheme IBE with a certain homomorphic property can be transformed to achieve key-insulated security, following [3].[1] We denote this construction by by KIS. Unfortunately, this scheme is insecure in case both user and base are corrupted (indeed, the scheme was not designed with this security property in mind).

Our results shows that FSE with a certain homomorphic property is sufficient to achieve intrusion resilience. Then, we may raise the natural question as to whether a generic PFSE scheme can be transformed to achieve intrusion resilience. Unfortunately, the answer seems to be "no" in general (at least using a "simple" construction as shown here) even if we assume that the key-update algorithm is appropriately homomorphic. More formally, any PFSE scheme which can be converted in this way can actually be cast as an FSE scheme anyway. We briefly discuss why. Intuitively, both the user and the base must share the secret key

[1] The construction in [3] is based on the Boneh-Franklin ID-based encryption scheme [4], but may be extended to use any ID-based encryption scheme with a certain homomorphic property.

of the PFSE scheme in order to achieve intrusion resilience. This requires that no single entity can have enough control to cause any security concerns. On the other hand, the user needs to decrypt a ciphertext. This indicates some separation between keys used for decryption and keys used for key updates. It would be interesting to formalize and rigorously prove the above informal reasoning.

This may raise another question of what level of security is achieved by using PFSE. We show that any primitive forward secure encryption scheme together with any secure ID-based encryption scheme that satisfies a certain homomorphic property can be transformed to achieve quasi-intrusion-resilience in Appendix B. The construction is called QISER, and the definition of quasi-intrusion-resilience is given in Appendix A. These abstraction of each security notion is shown in Table 1.

Remark: The Boneh-Franklin ID-based encryption scheme satisfies the necessary homomorphic property. Therefore, a forward-secure encryption scheme (e.g., [5]) combined with this IBE scheme satisfies quasi-intrusion-resilience.

Table 1. Abstraction of each security notion

	underlying notion	achieved security level
KIS[3]	IBE	key-insulated
QISER	PFSE + IBE	quasi-intrusion-resilient
FISER	FSE	intrusion-resilient

References

1. R. Anderson. "Two remarks on public-key cryptology." Invited Lecture, *ACM-CCCS '97*. Available at `http://www.cl.cam.ac.uk/ftp/users/rja14/forwardsecure.pdf`.
2. M. Bellare and S. K. Miner. "A forward-secure digital signature scheme." *Advances in Cryptology — Crypto '99*, LNCS vol. 1666, Springer-Verlag, 1999.
3. M. Bellare, and A. Palacio. "Protecting against key exposure: strongly key-insulated encryption with optimal threshold." Available at `http://eprint.iacr.org`.
4. D. Boneh and M. Franklin. "Identity based encryption from the Weil pairing." *Advances in Cryptology — Crypto 2001*, LNCS vol. 2139, Springer-Verlag, 2001. Full version to appear in *SIAM J. Computing* and available at `http://eprint.iacr.org/2001/090`.
5. R. Canetti, S. Halevi, and J. Katz, "A forward-secure public-key encryption scheme." *Advances in Cryptology — Eurocrypt 2003*, LNCS vol. 2656, Springer-Verlag, 2003.
6. Y. Dodis, M. Franklin, J. Katz, A. Miyaji and M. Yung. "Intrusion-resilient public-key encryption." *RSA — Cryptographers' Track 2003*, LNCS 2612, Springer-Verlag, 2003.

7. Y. Dodis, J. Katz, S. Xu, and M. Yung. "Key-insulated public-key cryptosystems." *Advances in Cryptology — Eurocrypt 2002*, LNCS vol. 2332, Springer-Verlag, 2002.
8. Y. Dodis, J. Katz, S. Xu, and M. Yung. "Strong key-insulated signature schemes." *Public Key Cryptography 2003*, LNCS vol. 2567, Springer-Verlag, 2003.
9. G. Itkis. "Intrusion-resilient signatures: generic constructions; or defeating a strong adversary with minimal assumptions." *Security in Communication Networks 2003*, LNCS vol. 2576, Springer-Verlag, 2002.
10. G. Itkis and L. Reyzin. "SiBIR: signer-base intrusion-resilient signatures." *Advances in Cryptology — Crypto 2002*, LNCS, vol. 2442, Springer-Verlag, 2002.

A Definition of Quasi-Intrusion Resilience

We introduce the notion of quasi intrusion resilience, which lies "in-between" key-insulated security and intrusion resilience. Informally, the security obtained is as follows: corrupting both the base and the user at the same time period means that any period before the first user corruption is secure; otherwise, repeated exposure of the user and the base only compromises those specific time periods during which the user's secret keys were exposed.

We generalize the notion of (t_0, Q)-compromise from Section 3.2 by considering two disjoint scenarios, *simultaneous* and *non-simultaneous* corruption. We call a corruption *simultaneous* if both user and base were compromised for the same time period and refresh period; otherwise we call the corruption *non-simultaneous*. More formally, we say the scheme is (t_0, Q)-simultaneous-compromised if one of the following is true:

- $sk_{t_0.r}$ is Q-exposed (for some r); or
- both $sk_{t'.r}$ and $skb_{t'.r}$ are Q-exposed (for some t' and r).

We say the scheme is (t_0, Q)-non-simultaneous-compromised if:

- $sk_{t_0.r}$ is Q-exposed (for some r); or
- both $sk_{t'.r}$ and $skb_{t'.r}$ are Q-exposed (for some r and $t' < t_0$); or
- both $sk_{t'.r}$ and $skb_{t'.r}$ are never both Q-exposed (for any r and $t' > t_0$).

One can consider definitions in which (t_0, Q)-simultaneous-compromise is disallowed, or in which (t_0, Q)-non-simultaneous-compromise is disallowed. A scheme secure against any adversary who does not (t_0, Q)-simultaneous-compromise the system is called non-simultaneous-compromise secure; the opposite case gives a system which is simultaneous-compromise secure. Obviously, an encryption scheme is (fully) intrusion resilient if and only if it is both simultaneous-compromise and non-simultaneous-compromise secure.

By slightly modifying the condition of (t_0, Q)-non-simultaneous-compromised, we may define a system as (t_0, Q)-quasi-non-simultaneous-compromised if:

- $sk_{t.r}$ is Q-exposed (for some r and $t \leq t_0$); or
- both $sk_{t'.r}$ and $skb_{t'.r}$ are Q-exposed (for some r and $t' < t_0$); or
- both $sk_{t'.r}$ and $skb_{t'.r}$ are never both Q-exposed (for any r and $t' > t_0$).

Let us define (t_0, Q)-quasi-non-simultaneous-compromised and CCA1 variation of intrusion-resilience as quasi-secure; that is, the adversary does not query after receiving the challenge ciphertext c from O_{LR} and the scheme is not (t_0, Q)-quasi-non-simultaneous-compromised. The notion of quasi-intrusion-resilience is given as follows.

Definition 3. *We say that a two-entity encryption scheme is quasi-intrusion-resilient against chosen ciphertext attacks (*`QIR-CCA`*) if it is intrusion resilient and also secure against quasi-non-simultaneous-compromise.*

B Generic Quasi-Intrusion-Resilient Encryption

B.1 Preparations

Our idea is to combine a primitive forward-secure encryption scheme with a secure ID-based encryption scheme, where key-extract algorithm has a *homomorphic-like* property. Let us define a homomorphic-like property of map,

$$\phi : G_1 \times S \to G_2,$$

where both G_1 and G_2 are groups and S is a set. The operation of G_1 and G_2 is represented additively, and S does not have to be a group. Then, we say that the map ϕ has a homomorphic-like property if for all $s_1, s_2 \in G_1$ and all $t \in S$

$$\phi(s_1 + s_2, t) = \phi(s_1, t) + \phi(s_2, t).$$

Now we give a general construction of quasi-intrusion-resilient scheme. Let S_2 and S_3 be sets, which are used in `PFSE`. We do not require any group property for `PFSE`. Let G_4 and G_5 be groups, which are used in `IBE`. The operations are represented additively.

• $\texttt{PFSE} = (\texttt{pfsKeyGen}, \texttt{pfsKeyUpd}, \texttt{pfsEnc}, \texttt{pfsDec})$:
⋄ $\texttt{pfsKeyGen}$: $\{0,1\}^* \times \mathbb{N} \to S_3 \times S_2$; $\texttt{pfsKeyGen}(k, T)=(\overline{sk}_0, \overline{pk})$
⋄ $\texttt{pfsKeyUpd} : S_3 \to S_3$; $\texttt{pfsKeyUpd}(\overline{sk}_t) = \overline{sk}_{t+1}$
⋄ $\texttt{pfsEnc} : S_2 \times \mathbb{N} \times \{0,1\}^n \to \{0,1\}^n$; $\texttt{pfsEnc}(\overline{pk}, t, M) = C$
⋄ $\texttt{pfsDec} : S_3 \times \{0,1\}^n \to \{0,1\}^n$; $\texttt{pfsDec}(\overline{sk}_t, C) = M$

ID-based encryption consists of key-generation, key-extraction, encryption, and decryption algorithms.
• $\texttt{IBE} = (\texttt{IBKeyGen}, \texttt{IBKeyExt}, \texttt{IBEnc}, \texttt{IBDec})$:
⋄ $\texttt{IBKeyGen}$: $\{0,1\}^* \to G_4 \times G_5$; $\texttt{IBKeyGen}(k)=(s_0, pk_1)$
Input: security parameter k
Output: master secret s_0, public key pk_1

⋄ $\texttt{IBKeyExt}$: $G_4 \times \mathbb{N} \to G_5$; $\texttt{IBKeyExt}(s_0, t) = isk_t$
Input: user ID t and secret s_0
Output: user secret key isk_t

`IBKeyExt` has to satisfy a homomorphic-like property for G_4 and G_5:

$$\texttt{IBKeyExt}(s_1 + s_2, t) = \texttt{IBKeyExt}(s_1, t) + \texttt{IBKeyExt}(s_2, t).$$

$\diamond$ `IBEnc`: $G_5 \times \mathbb{N} \times \{0,1\}^n \rightarrow \{0,1\}^n$; $\texttt{IBEnc}(pk_1, t, M) = C$
Input: public key pk_1, user ID t, message M
Output: cipher text C

$\diamond$ `IBDec`: $G_5 \times \{0,1\}^n \rightarrow \{0,1\}^n$; $\texttt{IBDec}(isk_t, C) = M$
Input: user secret key isk_t, ciphertext $C = \texttt{IBEnc}(pk_1, t, M)$
Output: message M

B.2 QISER

Let us describe the quasi-intrusion-secure encryption scheme `QISER` = (`KeyGen`, `BaseUpd`, `UserUpd`, `BaseRef`, `UserRef`, `Enc`, `Dec`). Here, user secret keys, user public keys, base secret keys, key update message, and key refresh message are defined on the following sets or groups:

$\diamond$ set of user public keys : $S_5 = S_2 \times G_5$
$\diamond$ set of user secret keys : $S_4 = S_3 \times G_4 \times G_5$
$\diamond$ group of base secret keys : G_4
$\diamond$ group of key update message : $G_6 = G_4 \times G_5$
$\diamond$ group of key refresh message : G_4

`KeyGen`: $\{0,1\}^* \times \mathbb{N} \rightarrow S_4 \times G_4 \times S_5$; $\texttt{KeyGen}(k, T)$=($sk_{0.0}$, $skb_{0.0}$, pk)
For inputs of security parameter k and time T,
1. Set $(s_0, pk_1) \leftarrow \texttt{IBKeyGen}(k)$ and $isk_0 \leftarrow \texttt{IBKeyExt}(s_0, 0)$.
2. Let s_0 be divided in $s_0 = sks_{0.0} + skb_{0.0}$ for randomly chosen $sks_{0.0} \in G_4$
3. Compute ($\overline{sk}_0$, $\overline{pk}$) $\leftarrow$ `pfsKeyGen`(k, T).
4. Set $pk = (\overline{pk}, pk_1)$ and $sk_{0.0} = (\overline{sk}_0, sks_{0.0}, isk_0)$.
5. Output $sk_{0.0}$, $skb_{0.0}$, and pk.

`BaseUpd`: $G_4 \rightarrow G_4 \times G_6$; $\texttt{BaseUpd}(skb_{t.r}) = (skb_{t+1.0}, sku_t)$
For an input of base secret key $skb_{t.r}$,
1. Compute $skb_{t+1.0} = skb_{t.r} - l_t$ for a random secret $l_t \in G_4$
2. Compute $u_t \leftarrow \texttt{IBKeyExt}(skb_{t+1.0}, t+1)$.
3. Output $skb_{t+1.0}$ and $sku_t = (l_t, u_t)$.

`UserUpd`: $S_4 \times G_6 \rightarrow S_4$; $\texttt{UserUpd}(sk_{t.r}, sku_t) = sk_{t+1.0}$
For inputs of user secret key $sk_{t.r} = (\overline{sk}_t, sks_{t.r}, isk_t)$ and update message $sku_t = (l_t, u_t)$,
1. Compute $sks_{t+1.0} = sks_{t.r} + l_t$.
2. Compute $isk_{t+1} = \texttt{IBKeyExt}(sks_{t+1.0}, t+1) + u_t$.
3. Compute $\overline{sk}_{t+1} \leftarrow \texttt{pfsKeyUpd}(\overline{sk}_t)$.
4. Output $sk_{t+1.0} = (\overline{sk}_{t+1}, sks_{t+1.0}, isk_{t+1})$.

`BaseRef`: $G_4 \rightarrow G_4 \times G_4$; $\texttt{BaseRef}(skb_{t.r}) = (skb_{t.r+1}, skr_{t.r})$
For an input of base secret key $skb_{t.r}$,
1. Compute $skb_{t.r+1} = skb_{t.r} - l_{t.r}$ for a random secret $l_{t.r} \in G_4$.
2. Output $skb_{t.r+1}$ and $skr_{t.r} = l_{t.r}$.

`UserRef`: $S_4 \times G_4 \rightarrow S_4$; $\texttt{UserRef}(sk_{t.r}, skr_{t.r}) = sk_{t.r+1}$
For inputs of user secret key $sk_{t.r} = (\overline{sk}_t, sks_{t.r}, isk_t)$ and refresh message $skr_{t.r}$,

1. Compute $sks_{t.r+1} = sks_{t.r} + skr_{t.r}$.
2. Output $sk_{t.r+1} = (\overline{sk}_t, sks_{t.r+1}, isk_t)$.

`Enc`: $S_5 \times \mathbb{N} \times \{0,1\}^n \rightarrow \{0,1\}^n$; $\texttt{Enc}(pk, t, M) = C$
For inputs of a public key pk, time t, and a message M,
1. Compute $C \leftarrow \texttt{IBEnc}(pk_1, t, \texttt{pfsEnc}(\overline{pk}, t, M))$.
2. Output C.

`Dec`: $S_4 \times \{0,1\}^n \rightarrow \{0,1\}^n$; $\texttt{Dec}(sk_{t.r}, C) = M$
For inputs of user secret key $sk_{t.r} = (\overline{sk}_t, sks_{t.r}, isk_{t.r})$ and a ciphertext C,
1. Compute $M \leftarrow \texttt{pfsDec}(\overline{sk}_t, \texttt{IBDec}(isk_{t.r}, C))$.
2. Output M.

B.3 Security Analysis

The following theorems will be proved in the final version.

Theorem 2. *Let A be an adversary of time complexity τ with at most Q queries to oracles $O \in \{O_{Dec}, O_{sec}, O_{LR}\}$ against* `QISER`*. If A has non-negligible advantage under non-simultaneous compromise, then there exists an adversary B performing a chosen ciphertext attack against the underlying* `IBE` *with at least the same advantage. The time complexity of B is at most $\tau + O(\log k)$, and the number of queries is at most Q.*

Theorem 3. *Let A be an adversary of time complexity τ with at most Q queries to oracles $O \in \{O_{Dec}, O_{sec}, O_{LR}\}$ against* `QISER`*. If A has non-negligible advantage under quasi-non-simultaneous-compromise, then there exists an adversary B performing a chosen ciphertext attack against the underlying* `PFSE` *with at least the same advantage. The time complexity of B is at most $\tau + O(\log k)$, and the number of queries is at most Q.*