A preliminary version of this paper appears in *Topics in Cryptology CT-RSA '09*, Lecture Notes in Computer Science Vol. ?? , M. Fischlin ed., Springer-Verlag, 2009. This is the full version.

Key Insulation and Intrusion Resilience Over a Public Channel

Mihir Bellare* Shanshan Duan† Adriana Palacio‡

Abstract

Key insulation (KI) and Intrusion resilience (IR) are methods to protect a user's key against exposure by utilizing periodic communications with an auxiliary helper. But existing work assumes a secure channel between user and helper. If we want to realize KI or IR in practice we must realize this secure channel. This paper looks at the question of how to do this when the communication is over what we are more likely to have in practice, namely a public channel such as the Internet or a wireless network. We explain why this problem is not trivial, introduce models and definitions that capture the desired security in a public channel setting, and provide a complete (and surprising) answer to the question of when KI and IR are possible over a public channel. The information we provide is important to guide practitioners with regard to the usage of KI and IR and also to guide future research in this area.

*Dept. of Computer Science & Engineering, University of California at San Diego, 9500 Gilman Drive, La Jolla, California 92093, USA. E-Mail: mihir@cs.ucsd.edu. URL: http://www-cse.ucsd.edu/users/mihir. Supported in part by NSF grant CNS 0524765 and CNS 0627779 and a gift from Intel corporation.

†Dept. of Computer Science & Engineering, University of California at San Diego, 9500 Gilman Drive, La Jolla, California 92093, USA. E-Mail: shduan@cs.ucsd.edu. URL: http://www-cse.ucsd.edu/users/shduan. Supported in part by the grants of the first author.

‡Computer Science Department, Bowdoin College, 8650 College Station, Brunswick, ME 04011-8486, USA. E-Mail: apalacio@bowdoin.edu. URL: http://academic.bowdoin.edu/faculty/A/apalacio/.

Contents

1 Introduction

Key Insulation (KI) [15, 16] and Intrusion Resilience (IR) [13, 20] are technologies to protect against key exposure. They have been extensively researched in the cryptographic community and we have lots of schemes, variations and extensions [13, 14, 15, 16, 18, 19, 20]. However, all this work assumes a secure communication channel between the parties. If we want to realize KI or IR in practice we must realize this secure channel. How can this be done? Surprisingly, this fundamental question has received no attention until now. We address it and turn up some surprising answers which have important implications for the realizability of KI and IR in practice.

1.1 Background

An important threat to the security of cryptography-using applications is exposure of the secret key due to viruses, worms or other break-ins allowed by operating-system holes. Forward security [1, 2, 7, 10] is one way to counter this, or at least mitigate the damage caused. Here the user has a single, fixed public key pk whose lifetime is divided into stages $1, \ldots, N$. The secret (signing or decryption) key evolves with time: at the start of stage i, the user computes its stage i secret key usk_i as a function of its stage $i-1$ secret key usk_{i-1} and then discards the latter. The security condition is that for $j < i$, a break-in during stage i (resulting in exposure of usk_i) does not allow the adversary to compute usk_j or compromise its uses. (Meaning that forgery of documents with date j or decryption of ciphertexts sent in stage j remains hard.) Once usk_i is exposed, however, $usk_{i+1}, \ldots, usk_N$ are automatically compromised (they can be computed from usk_i), and the best the user can hope to do about this is detect the break-in and revoke the public key.

Key-insulated (KI) security as introduced by Dodis, Katz, Xu, and Yung [15, 16] and refined by [4] attempts to provide both forward and backward security, meaning a break-in during stage i leaves usk_j uncompromised for all $j \neq i$. More generally, break-ins for all stages $i \in I$ leave usk_j and its uses secure for all $j \in [N] \setminus I$, where $[N] = \{1, \ldots, N\}$. To accomplish this, an auxiliary party, called a helper, is introduced. The secret key usk_i of stage i is now computed by the user not merely as a function of usk_{i-1}, but also of a key hsk_i sent by the helper to the user at the start of stage i. The advantage of this system (over a merely forward-secure one) is that the public key is *never* revoked. Intrusion resilience (IR) [20, 13] is an extension where forward and backword security are provided even if both user and helper are compromised as long as the compromise is not simultaneous and, even in the latter case, forward security is assured. Further extensions and variants include KI with parallel helpers [18] and KI (hierarchical) identity-based encryption [19]. Our discussion below will focus on the simpler KI case. We will discuss the extension to IR later.

KI security requires that the communication channel between user and helper is secure. Indeed, if not, meaning if an adversary could obtain the helper keys $hsk_1, \ldots, hsk_N$ sent over the channel, a single break-in in a stage i would allow it to compute *all* subsequent user secret keys by simply using the key-update process of the user, and KI would end up providing no more than forward security, which does not even need a helper. In previous works, this secure-channel assumption is built into the model, which denies the adversary hsk_j unless it has broken in during stage j.

1.2 Realizing the secure channel

To deploy KI in practice we must have some way to realize the secure channel. In some settings it may be possible to do this through physical means, but such settings are rare. The range of application for KI would be greatly increased if the communication between user and helper could flow over a public channel such as the Internet or a wireless network. This would allow the helper

to be, for example, a server on the Internet. Alternatively, the helper could be your cell phone with the user being your laptop. (In this case, even though the devices may be in close proximity, the communication would be over a public wireless phone network.)

While definitely important for applications, enabling KI over public channels looks at first to be something trivial. This is because we would appear to know very well how to implement a secure channel over a public one. After all, isn't this the main task of basic cryptography? Specifically, let us just use encryption and authentication, either under a symmetric key shared by the parties, or under public keys.

However, we make the important observation that this standard solution runs into an inherent problem here, where the name of the game is break-in and key exposure. Namely, if the adversary breaks in during some stage i, one should realistically assume it exposes not just usk_i but also any keys used to secure the channel. (Meaning either the shared key or the user's decryption key.) This renders the channel insecure from then on, and key-insulated security vanishes (more accurately, one has only forward security) as explained above.

The above indicates that realizing KI over a public channel is nontrivial but not (yet) that it is impossible. The reason is that we have not yet exploited the full power of the model. Specifically there are two capabilities one can offer the parties. First, since we are already in a setting where keys evolve, instead of trying to secure the channel with static keys, we could allow channel-securing keys to evolve as well. Second, we could allow the update process to be an interactive protocol rather than merely a single flow.

1.3 Our model

What the above reflects is that we need a new model to formally investigate the possibility of KI over a public channel. Providing such a model is the first contribution to our paper. In our model, the user in stage i has (in addition to usk_i) a stage i channel-securing key uck_i, while the helper has a corresponding hck_i. At the start of stage $i+1$, the parties engage in an arbitrary interactive *channel-update* protocol. This protocol uses —and aims to get its security from— the current channel keys uck_i, hck_i. Its goal is two-fold: to (securely) communicate hsk_{i+1} from helper to user, and to "refresh" the channel keys, meaning deal the helper with a new key hck_{i+1} and the user with a corresponding new key uck_{i+1}. Once the protocol terminates, the user can update usk_i to usk_{i+1} using hsk_{i+1} as before, install uck_{i+1} as its new channel key, and discard both usk_i and uck_i. As an example, the protocol could begin with an authenticated session-key exchange based on its current channel keys and then use the session key to securely transfer hsk_{i+1} and fresh channel keys. But now, a break-in during period i exposes not only usk_i and hsk_i but also uck_i. Actually we go further, allowing the adversary to even obtain the user coins underlying the stage i channel-update protocol execution. This is realistic because the intruder could be on the system when the protocol executes, but this added adversary capability will make our proofs harder. While the core elements of the new model are natural and clear, there are subtle details. In Section 4, we describe our model and provide a formal definition of KI security over a public channel.

1.4 Our results

Now that we have a model, we ask whether it is possible to design KI schemes secure in this public channel model. Interestingly, the answer turns out to depend on whether the adversary is active or merely passive. Specifically, the answer is "no" in the first case and "yes" in the second. Let us now elaborate on these results.

ACTIVE SECURITY. The communication security model cryptographers prefer to consider is that of an active adversary who has full control of the channel. It can not only see all transmissions, but stop, inject or alter any transmission. This is the model adopted, for example, in the work of Canetti and Krawczyk defining notions of secure channels [11, 12], and also in work on session-key exchange [5, 11]. It would be desirable to achieve public-channel KI security in the face of such an adversary. We show that this is impossible. That is, even in our above-described model, which allows an interactive channel-update protocol and evolving channel-security keys, an active adversary can always succeed in breaking the scheme. The reason is that after it breaks in, it obtains the user's channel-security key and can thus impersonate the user. We note that authentication (such as an authenticated session-key exchange) does not prevent this since the adversary acquires all the user's credentials via the break-in. This negative result is particularly strong because our public-channel KI model is as generous as one can get, while keeping in the spirit of KI.

There seem to be only two ways to circumvent the negative result. The first is to revoke the public key upon break-in discovery, but if one is willing to do this, one may as well just use forward security and avoid the helper altogether. Indeed, the whole point of the helper and KI is to never have to revoke the public key. The other possibility is to use an out-of-band method to redistribute channel-securing keys after break-in discovery such as a physically secure channel. But this is just an assumed secure channel under another name, exactly what we are trying to avoid. In conclusion, our result suggests that it would be inadvisable to implement any form of KI when the channel may be open to active attack.

PASSIVE SECURITY. On the positive side, we show that public-channel KI is possible against an adversary that is allowed only a passive attack on the communication channel. (Meaning it can eavesdrop, but not inject messages.) Our method is general, meaning it yields a compiler that can take any KI scheme secure in the secure-channel model and turn it into a KI scheme secure in our public-channel model under passive attack. The transformation is simple. Our channel-update protocol begins with a secure key exchange (e.g., Diffie-Hellman) to get a session key under which the helper encrypts the data it needs to transmit. The key exchange is not authenticated: this is not necessary for security against passive attack and, given the above, would not help to achieve security against active attack. We clarify that our choice of channel-update protocol is purely illustrative. The reader can surely think of others that will work.

This positive result is significant for two reasons. First, it shows that KI is at least possible over a channel where the adversary may be able to eavesdrop but finds it hard to inject or corrupt transmissions. Second, the result shows that our new method, allowing an interactive channel-update protocol, has borne fruit. Indeed, even KI under passive attack is not possible when the communication consists of a single transmission from helper to user.

Although the protocol is simple, there are subtleties in the proof arising from the strength of our model which allows the adversary to obtain the user coins from the channel-update protocol execution in any stage in which it breaks in. A consequence of this is that the starting secure-channel KI scheme needs to have optimal threshold, meaning be secure even if there are break ins in all but one stage. Some early secure-channel KI encryption schemes [15] were threshold and did not have this property, and, in this case, we cannot offer security over a public channel even in the presence of a passive adversary. Luckily, secure-channel KI schemes with optimal threshold exist for both encryption [4] and signatures [16].

PRACTICAL IMPLICATIONS. Our results imply that KI will only work if one has a channel whose physical properties preclude active attack. Anyone contemplating actual usage of KI needs to be aware of this limitation and the need to be careful about the choice of channel.

1.5 Extensions

The intrusion resilience (IR) setting of [20, 13] continues to make the secure-channel assumption, and our results extend to it. However the model is considerably more complex due to the presence of both refreshes and updates and again there are subtle details to be careful about in creating the public channel analog. We recall the secure-channel IR model in Section 7 and then provide a detailed description of our public-channel IR model in Section 8. When this is done, the negative result, showing the impossibility of IR over a public channel in the presence of an active adversary, carries over easily from the KI case. We need to extend the previous positive result, however. We are able to show that secure key exchange can still be used for both refresh and update to transform any secure-channel IR scheme into a public-channel IR scheme secure against passive adversaries. The proof is, however, more complex than in the KI case and is given in Section 9. Similar extensions hold for the many variant notions in this area, including strong KI security [15, 16] and KI with parallel helpers [18].

1.6 Discussion

Cryptographic protocols commonly make the assumption that parties are connected by secure channels. This abstraction would seem both natural and convenient; after all, isn't this exactly what standard cryptography (encryption and authentication) gives us? Yet there are settings where secure channels are surprisingly difficult to realize. One example is secure computation, where a secure channel between each pair of parties is a standard assumption [8]. Yet this channel is astonishingly difficult to realize, at least in the public-key setting, due in part to the selective-decryption problem [17]. Solutions were finally given by [9]. Our work provides another example.

2 Definitions

We let $\mathbb{N} = \{1, 2, \ldots\}$ be the set of positive integers, and for $N \in \mathbb{N}$ we let $[N] = \{1, \ldots, N\}$. The empty string is denoted ε. The notation $x \xleftarrow{\$} S$ denotes that x is selected randomly from set S. Unless otherwise indicated, an algorithm may be randomized. An adversary is an algorithm. If A is an algorithm, then the notation $x \xleftarrow{\$} A(a_1, a_2, \ldots)$ denotes that x is assigned the outcome of the experiment of running A on inputs $a_1, a_2, \ldots$, with fresh coins. If A is deterministic, we might write $x \leftarrow A(a_1, a_2, \ldots)$ instead.

Games. We will use code-based games [6] in definitions and some proofs. We recall some background here. A game —see Figure 1 for an example— has an **Initialize** procedure, procedures to respond to adversary oracle queries, and a **Finalize** procedure. A game G is executed with an adversary A as follows. First, **Initialize** executes and its outputs are the inputs to A. Then, A executes, its oracle queries being answered by the corresponding procedures of G. When A terminates, its output becomes the input to the **Finalize** procedure. The output of the latter, denoted G^A, is called the output of the game, and we let "$\mathrm{G}^A \Rightarrow y$" denote the event that this game output takes value y. Variables not explicitly initialized or assigned are assumed to have value $\perp$, except for booleans which are assumed initialized to false.

Interactive algorithms. We will model each party in a two-party protocol as an *interactive algorithm*. Such an algorithm I takes as input an *incoming message* M_{in}, a *current state* St, and a *decision* d which can be acc, rej or $\perp$. Its output, denoted $I(M_{\mathrm{in}}, St, d)$, is a triple $(M_{\mathrm{out}}, St', d')$ consisting of an *outgoing message*, an updated state, and an updated decision. We require that if $d \neq \perp$ then $M_{\mathrm{out}} = \perp$, $St' = St$, and $d' = d$. Our convention is that the initial state provided to

an interactive algorithm is its local input and random coins. Given a pair of interactive algorithms (I, J), we assume that the first move in the interaction always belongs to I. The first incoming message for I is set to ε. An interactive algorithm terminates when its decision becomes acc or rej. Once it terminates, it outputs $\perp$ as its outgoing message in response to any incoming message and its state and decision stay the same. The *local output* of an interactive algorithm is its final state.

Given a pair of interactive algorithms (I, J) with local inputs x_I, x_J and coins ω^I, ω^J respectively, we define $\mathbf{Run}(I, x_I, J, x_J; \omega^I, \omega^J)$ to be the quintuple $(\mathrm{Conv}, St_I, d_I, St_J, d_J)$ consisting of the conversation transcript (meaning the sequence of messages exchanged between the parties), I's local output, I's decision, J's local output, and J's decision, respectively, after an interaction in which I has local input x_I and random coins ω^I and J has local input x_J and random coins ω^J. We let $\mathbf{Run}(I, x_I, J, x_J)$ be the random variable whose value is $\mathbf{Run}(I, x_I, J, x_J; \omega^I, \omega^J)$ when ω^I, ω^J are chosen at random.

3 Key insulation in the secure-channel model

We will take a modular approach to KI over a public channel, where a public-channel KI scheme consists of a (standard) secure-channel KI scheme —meaning one in the model of an assumed-secure channel— together with a channel-key-update protocol. We will then be able to give "compiler" style results which transform any secure-channel KI scheme into a public-channel KI scheme for suitable channel-key-update protocols. (Of course, this is only for passive adversaries since in the active case we will show that KI over public channels is impossible.) To enable this we first recall a definition of secure-channel KI. The latter has been defined for both encryption [15, 4] and signatures [16]. For simplicity, we will treat the case of signatures. The case of encryption is entirely analogous and all our results carry over. Our definition below differs from that of [16] in some details, but this does not affect the results.

A *key-updating signature scheme* $\mathsf{KUS} = (\mathsf{KG}, \mathsf{HKU}, \mathsf{UKU}, \mathsf{Sig}, \mathsf{Ver})$ is specified by five algorithms with the following functionality. The randomized *key-generation algorithm* KG returns (pk, usk_0, hsk), where pk is the user public key, usk_0 is the *stage 0 user secret key*, and hsk is the *master helper key*. The user is initialized with pk, usk_0, while the helper is initialized with pk, hsk. At the start of stage $a \geq 1$, the helper applies the deterministic *helper key-update algorithm* HKU to a, pk, hsk to obtain a *stage a helper key* hsk_a, which is then assumed to be conveyed to the user via a secure channel. The user receives hsk_a from the helper and then applies the deterministic *user key-update algorithm* UKU to a, pk, hsk_a, usk_{a-1} to obtain the *stage a user secret key* usk_a. The user then discards (erases) usk_{a-1}. In stage a the user can apply the *signing algorithm* Sig to a, its stage a secret key usk_a, and a message $M \in \{0,1\}^*$ to obtain a pair (a, σ), consisting of the stage number a and a signature σ. During stage a anyone can apply the deterministic *verification algorithm* Ver to pk, a message M, and a pair (i, σ) to obtain either 1, indicating acceptance, or 0, indicating rejection. We require that if (i, σ), where $1 \leq i \leq a$, was produced by applying the signing algorithm to i, usk_i, M then $\mathsf{Ver}(pk, M, (i, \sigma)) = 1$.

SECURITY. Consider game KIS of Figure 1. The **Initialize** procedure provides adversary A with input pk. A can call its **Next** oracle to move the system into the next stage. It may break in during the current stage by calling its **Expose** oracle and getting back the user and helper keys for that stage. A may obtain signatures for messages of its choice during the current stage by calling its **Sign** oracle. To win, A must output a message M and a signature (j, σ) such that j is an unexposed stage, $\mathsf{Ver}(pk, M, (j, \sigma)) = 1$, and M was not queried to **Sign** during stage j. A's

procedure Initialize
$(pk, usk_0, hsk) \xleftarrow{\$} \mathsf{KG}$; $a \leftarrow 0$; $S \leftarrow \emptyset$; $E \leftarrow \emptyset$
Return pk

procedure Next()
$a \leftarrow a + 1$
$hsk_a \leftarrow \mathsf{HKU}(a, pk, hsk)$
$usk_a \leftarrow \mathsf{UKU}(a, pk, hsk_a, usk_{a-1})$

procedure Finalize$(M, (j, \sigma))$
Return $(j \notin E \wedge (j, M) \notin S \wedge \mathsf{Ver}(pk, M, (j, \sigma)) = 1)$

procedure Expose()
$E \leftarrow E \cup \{a\}$
Return (usk_a, hsk_a)

procedure Sign(M)
$(a, \sigma) \xleftarrow{\$} \mathsf{Sig}(a, usk_a, M)$
$S \leftarrow S \cup \{(a, M)\}$
Return (a, σ)

Figure 1: Game KIS used to define KI signatures in the secure-channel model.

advantage is

$$\mathbf{Adv}^{\mathrm{ki}}_{\mathsf{KUS}}(A) \;=\; \Pr\left[\,\mathsf{KIS}^{A} \Rightarrow \mathsf{true}\,\right].$$

We adopt the convention that the *running time* of an adversary A is the execution time of the entire game, including the time taken for initialization, the time taken by the oracles to compute replies to the adversary's queries, and the time taken for finalization.

The implicit secure-channel assumption. As discussed in Section 1, the secure-channel assumption is implicit in the above model. This is due to the fact that A is *not* given hsk_a for stages a in which it did not make an **Expose** query. Also note that the assumption is necessary, for if A had an additional oracle **Get** that returned hsk_a, but the rest of the game was the same, it could win via

$\mathbf{Next}()$; $(hsk_1, usk_1) \leftarrow \mathbf{Expose}()$; $\mathbf{Next}()$
$hsk_2 \leftarrow \mathbf{Get}()$; $usk_2 \leftarrow \mathsf{UKU}(2, pk, hsk_2, usk_1)$
$(2, \sigma) \leftarrow \mathsf{Sig}(2, usk_2, 0)$; return $(0, (2, \sigma))$

4 Key insulation in the public-channel model

We saw above that a secure channel between helper and user is both assumed and necessary in the existing notion of KI. Here we consider how the channel can be implemented. Let us first discuss how key exposure implies failure of the obvious way to secure the channel.

Static keys won't secure the channel. The obvious solution is to use standard cryptography. Let the helper have a signing key sk whose corresponding verification key vk is held by the user, and correspondingly, let the user have a decryption key dk whose corresponding encryption key ek is held by the helper. (These keys are generated and distributed honestly and securely along with usk_0, hsk when the system is initialized. The cryptography could be symmetric or asymmetric. In the first case, the signature is a MAC and encryption is symmetric, so that $sk = vk$ and $dk = ek$. In the second case, the signature and encryption are public-key based.) Now in stage a, the helper sends (C, σ) to the user, where C is an encryption of hsk_a under ek and σ is a signature of C under sk. The user verifies the signature using vk and decrypts C using dk to get hsk_a. This, however, fails completely to provide security in the key-exposure setting, even for an adversary that is merely passive with regard to channel access. (That is, it can eavesdrop the

communication but not send messages itself.) This is because one must realistically assume that a break-in in a stage a exposes all information the user has, which includes not only usk_a but also dk. Equipped with usk_a, dk via the break-in, the adversary can now obtain the stage $a+1$ channel transmission (C_{a+1}, σ_{a+1}) via its channel access, decrypt C_{a+1} using dk to get hsk_{a+1}, and compute $usk_{a+1} = \mathsf{UKU}(a+1, pk, hsk_{a+1}, usk_a)$. Continuing in this fashion, it can obtain usk_i for all $i \geq a$.

Evolving channel-securing keys. The above is already something of which potential implementers should be aware, but not yet enough to give up hope of obtaining KI, for there is an obvious next step, which we take. Namely, let us allow the channel to be secured not under keys that are static but which themselves evolve, so that a break-in exposes only the current keys. This section introduces and formalizes a very general model to this end, where an interactive protocol (such as a secure key exchange) may be used in each step to provide a secure channel and also update the channel keys.

Public-channel key updating signature schemes. A *public-channel key-updating signature scheme* is a triple $\mathsf{PCKUS} = (\mathsf{KUS}, \mathsf{CKG}, (\mathsf{U}, \mathsf{H}))$, where $\mathsf{KUS} = (\mathsf{KG}, \mathsf{HKU}, \mathsf{UKU}, \mathsf{Sig}, \mathsf{Ver})$ is a key-updating signature scheme, CKG is the channel-key-generation algorithm, and the channel-key-update protocol (U, H) is a pair of interactive algorithms to be run by user and helper, respectively. Let us now explain how the system runs.

Algorithm CKG returns (uck_0, hck_0), where uck_0 is the *stage 0 user channel key* and hck_0 is the *stage 0 helper channel key*. When the user is initialized, in addition to the public key pk and stage 0 user secret key usk_0 produced by KG, the user is given uck_0. When the helper is initialized, in addition to pk and the master helper key hsk (also generated by KG), the helper is given hck_0.

In any stage a ($a \geq 0$), the user holds not only its stage a user secret key usk_a, but also a stage a user channel key uck_a. The helper holds hsk and a stage a helper channel key hck_a. At the start of stage $a+1$, the helper computes $hsk_{a+1} = \mathsf{HKU}(a+1, pk, hsk)$. The parties then engage in the channel-key-update protocol (U, H). The local input of U is the stage a user secret key usk_a, the stage a user channel key uck_a and some random coins ω_a^U, while the local input of H is the stage $a+1$ helper key hsk_{a+1}, the stage a helper channel key hck_a and some random coins ω_a^H. After the interaction, the expected local output of U is hsk_{a+1} plus the stage $a+1$ user channel key uck_{a+1}, while the expected local output of H is the stage $a+1$ helper channel key hck_{a+1}. Once the protocol has completed, the user can update its key as before, namely it computes $usk_{a+1} = \mathsf{UKU}(a+1, pk, hsk_{a+1}, usk_a)$. It then discards not only usk_a but also its previous channel key uck_a. We require the natural correctness condition, namely that the stage $a+1$ helper key produced by U in the interaction in which U has input usk_a, uck_a, ω_a^U and H has input $hsk_{a+1}, hck_a, \omega_a^H$, is hsk_{a+1} with probability one. In addition, we require that at the end of the interaction, U's decision d_{a+1}^U and H's decision d_{a+1}^H are both acc.

Security. We proceed to formalize two notions of security for public-channel key-updating signature schemes: key insulation under active and passive attacks. We first provide definitions and then explanations. Let $\mathsf{PCKUS} = ((\mathsf{KG}, \mathsf{HKU}, \mathsf{UKU}, \mathsf{Sig}, \mathsf{Ver}), \mathsf{CKG}, (\mathsf{U}, \mathsf{H}))$ be a public-channel key-updating signature scheme. We consider an adversary A interacting with the games of Figure 2. The **Initialize** procedure gives A input pk. In an *active attack*, A is provided with oracles **Next**, **Expose**, **SendU**, **SendH**, and **Sign**, while in a *passive attack* it is provided with oracles **Next**, **Expose**, **Conv**, and **Sign**. It may query the oracles adaptively, in any order it wants, with the following restriction: In the case of an active adversary, as soon as **SendU** returns $d_{a+1}^U = \mathsf{acc}$ and **SendH** returns $d_{a+1}^H = \mathsf{acc}$, A makes a query to oracle **Next**. In the case of a passive adversary, every query to oracle **Conv** is immediately followed by a query to oracle **Next**. Eventually, A outputs a message M and a signature (j, σ) and halts. An active (resp., passive) adversary

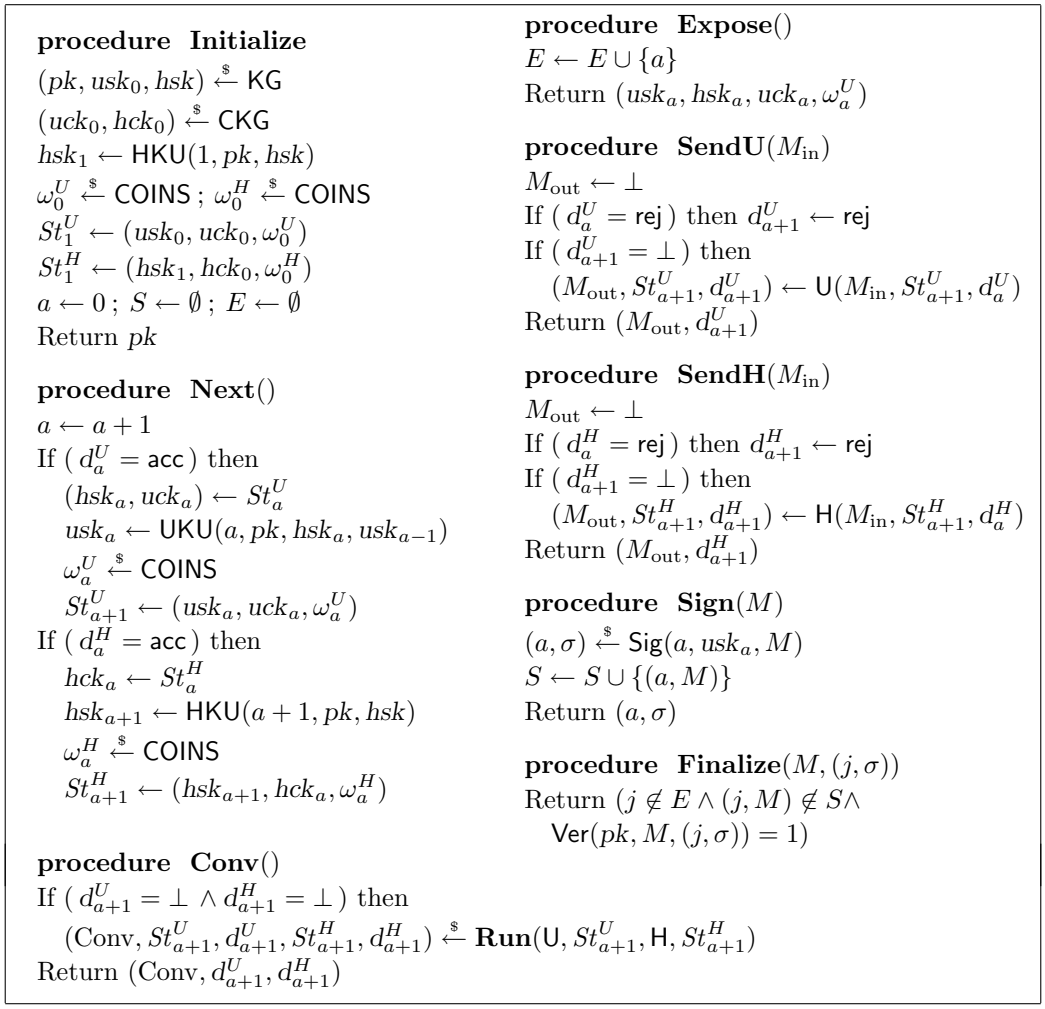

Figure 2: Games used to define public-channel key insulation under active and passive attack. Game PCKI-aa includes all of the procedures except **Conv**, while game PCKI-pa includes all except **SendU** and **SendH**.

is said to win if game PCKI-aa (resp., PCKI-pa) returns true, meaning j is an unexposed stage, $\mathsf{Ver}(pk, M, (j, \sigma)) = 1$, and M was not queried to oracle **Sign** during stage j. For $\mathsf{atk} \in \{\mathsf{aa}, \mathsf{pa}\}$, A's atk-advantage is

$$\mathbf{Adv}^{\text{pcki-atk}}_{\mathsf{PCKUS}}(A) \;=\; \Pr\left[\,\mathsf{PCKI\text{-}atk}^{A} \Rightarrow \mathsf{true}\,\right].$$

Again, we adopt the convention that the *running time* of an adversary A is the execution time of the entire game, including the time taken for initialization, the time taken by the oracles to compute replies to the adversary's queries, and the time taken for finalization.

Explanation. An active adversary has full control over the communication between the helper and the user. It can deliver messages out of order, modify messages or inject messages of its own choosing. This is modeled by providing the adversary access to oracles **SendU** and **SendH**, which represent the user and helper, respectively, running the channel-key-update protocol. Once this protocol terminates, the adversary is required to call its **Next** oracle to move the system into the next stage. This models the user updating his keys as soon as he obtains the helper secret key for the next stage. As in the case of key insulation in the secure-channel model, the adversary may break in during the current stage by calling its **Expose** oracle, but here it gets back the user secret key, the helper key, the user channel key, and the user's coins, for that stage. As before, the adversary may obtain signatures for messages of its choice during the current stage by calling its **Sign** oracle. To win, it must output a valid forgery for an unexposed stage.

A passive adversary cannot modify or inject messages, but it can eavesdrop on the communication channel, obtaining transcripts of conversations between the user and the helper. We model this by providing the adversary access to oracle **Conv** which runs the channel-key-update protocol and returns the conversation transcript and the decisions of U and H. In all other respects, a passive adversary is like an active adversary: as soon as the channel-key-update protocol terminates, the adversary is required to call its **Next** oracle to move the system into the next stage, the adversary can break in during the current stage, it can obtain signatures for messages of its choice during the current stage, and its goal is to produce a valid forgery for an unexposed stage.

5 Impossibility of public-channel KI under active attack

We show that the notion of public-channel key insulation under active attack is unachievable, meaning all public-channel key-updating signature schemes are vulnerable to an active attack. The precise statement of our result is the following.

Theorem 5.1 Let $\mathsf{PCKUS} = ((\mathsf{KG}, \mathsf{HKU}, \mathsf{UKU}, \mathsf{Sig}, \mathsf{Ver}), \mathsf{CKG}, (\mathsf{U}, \mathsf{H}))$ be a public-channel key-updating signature scheme. Let t_{KG}, t_{HKU}, t_{UKU}, t_{Sig}, t_{Ver}, and t_{CKG} denote the running times of the corresponding algorithms, and $t_{(\mathsf{U},\mathsf{H})}$ denote the running time of protocol (U, H). Let m be the maximum number of moves in this protocol. Then there exists an adversary A against PCKUS that makes one query to oracle **Next**, one **Expose** query, at most $\lceil m/2 \rceil$ **SendU** queries, at most $2\lceil m/2 \rceil$ **SendH** queries, and no **Sign** queries, such that

$$\mathbf{Adv}^{\text{pcki-aa}}_{\mathsf{PCKUS}}(A) \;=\; 1\,.$$

Furthermore, the running time of A is $t_{\mathsf{KG}} + t_{\mathsf{CKG}} + 2t_{\mathsf{HKU}} + 2t_{\mathsf{UKU}} + 2t_{(\mathsf{U},\mathsf{H})} + t_{\mathsf{Sig}}$.

The proof of the above theorem is simple, as is not uncommon for impossibility results, where the key insights are in the development of the model and the question posed.

Proof: A is defined in Figure 3.

Adversary $A(pk)$
$M_{\text{in}} \leftarrow \varepsilon$; $dec_1^U \leftarrow \perp$; $dec_1^H \leftarrow \perp$
While ($dec_1^U = \perp \vee dec_1^H = \perp$) do
$(M_{\text{out}}, dec_1^U) \xleftarrow{\$} \mathbf{SendU}(M_{\text{in}})$; $(M_{\text{in}}, dec_1^H) \xleftarrow{\$} \mathbf{SendH}(M_{\text{out}})$
$\mathbf{Next}()$; $(usk_1, hsk_1, uck_1) \leftarrow \mathbf{Expose}()$
$St_2^U \leftarrow uck_1$; $M_{\text{in}} \leftarrow \varepsilon$; $dec_2^U \leftarrow \perp$; $dec_2^H \leftarrow \perp$
While ($dec_2^U = \perp \vee dec_2^H = \perp$) do
$M_{\text{out}} \leftarrow \perp$
If ($dec_1^U = \mathsf{rej}$) then $dec_2^U = \mathsf{rej}$
If ($dec_2^U = \perp$) then $(M_{\text{out}}, dec_2^U, St_2^U) \xleftarrow{\$} \mathsf{U}(M_{\text{in}}, St_2^U, dec_1^U)$
$(M_{\text{in}}, dec_1^H) \xleftarrow{\$} \mathbf{SendH}(M_{\text{out}})$
$(hsk_2, uck_2) \leftarrow St_2^U$; $usk_2 \leftarrow \mathsf{UKU}(2, pk, hsk_2, usk_1)$; $(2, \sigma) \xleftarrow{\$} \mathsf{Sig}(2, usk_2, 0)$
Return $0, (2, \sigma)$

Figure 3: Adversary A for the proof of Theorem 5.1.

First A simulates the execution of the channel-key-update protocol using oracles **SendU** and **SendH** by delivering messages faithfully between them, starting with message ε delivered to **SendU**. Then A makes a query to oracle **Next** which results in the computation of the stage 1 user secret key usk_1. It then makes a query to **Expose** obtaining $(usk_1, hsk_1, uck_1, \omega_1^U)$. Setting $St_2^U = (usk_1, uck_1, \omega_1^U)$, A then plays the role of the user in an interaction with oracle **SendH**, by computing U itself. At the end of the interaction, A obtains hsk_2, uck_2. It can then compute usk_2 by applying the user key-update algorithm UKU. Finally, it computes a valid signature $(2, \sigma)$ for message 0 by applying the signing algorithm Sig with user secret key usk_2, and it outputs a forgery $0, (2, \sigma)$. A clearly satisfies the claims in the theorem. ∎

6 Possibility of public-channel KI under passive attack

Given a KI signature scheme in the secure-channel model, we show in this section how to transform it into a KI signature scheme secure against passive attack in the public-channel model. We first discuss the primitives we use, namely an arbitrary secret-key-exchange protocol and an arbitrary one-time symmetric encryption scheme.

Secret-key-exchange (SKE) Protocol. An SKE protocol with key length k is a pair of interactive algorithms (I, J) each of which has local output a k bit string. We require that

$$\Pr\left[(\text{Conv}, K_I, d^I, K_J, d^J) \xleftarrow{\$} \mathbf{Run}(\mathsf{I}, \varepsilon, \mathsf{J}, \varepsilon) \ : \ (K_I = K_J) \wedge (d^I = d^J = \mathsf{acc}) \right] = 1 \ ,$$

meaning the parties agree on a common key. For security we require that the common key be computationally indistinguishable from random. This is captured by defining the ske-advantage of an adversary A as

$$\mathbf{Adv}^{\text{ske}}_{(\mathsf{I},\mathsf{J})}(A) \ = \ 2 \cdot \Pr\left[\mathsf{SKE}^A_{(\mathsf{I},\mathsf{J})} \Rightarrow \mathsf{true} \right] - 1 \ ,$$

where game $\mathsf{SKE}_{(\mathsf{I},\mathsf{J})}$ is defined in Figure 4.

One example of a suitable SKE protocol is a Diffie-Hellman key exchange. (The DH key needs to be suitably hashed to a k bit string.) Another possibility, based on any asymmetric encryption

procedure Initialize $b \xleftarrow{\$} \{0,1\}$ **procedure Conv**() $(\text{Conv}, K_1, d^I, K_1, d^J) \xleftarrow{\$} \mathbf{Run}(\mathsf{I}, \varepsilon, \mathsf{J}, \varepsilon)$ $K_0 \xleftarrow{\$} \{0,1\}^k$ Return $(\text{Conv}, d^I, d^J, K_b)$ **procedure Finalize**(d) Return $(b = d)$	**procedure Initialize** $K \xleftarrow{\$} \{0,1\}^k$ $b \xleftarrow{\$} \{0,1\}$ **procedure LR**(M_0, M_1) $C \xleftarrow{\$} \mathsf{SEnc}(K, M_b)$ Return C **procedure Finalize**(d) Return $(b = d)$

Figure 4: Game $\mathsf{SKE}_{(\mathsf{I},\mathsf{J})}$ on the left is used to define security of SKE protocol (I, J) and game $\mathsf{INDCPA}_{\mathsf{SE}}$ on the right is used to define security of symmetric encryption scheme $\mathsf{SE} = (\mathsf{SEnc}, \mathsf{SDec})$. In both cases, the key length is k.

scheme $(\mathsf{AKg}, \mathsf{AEnc}, \mathsf{ADec})$, works as follows. I picks a public/secret key pair (pk, sk) by running AKg and sends pk to J. The latter selects a random k-bit string K, encrypts it under pk using AEnc and sends the ciphertext to I. I decrypts the ciphertext with sk using ADec to obtain K.

Symmetric encryption. A symmetric encryption scheme $\mathsf{SE} = (\mathsf{SEnc}, \mathsf{SDec})$ with key length k consists of two algorithms. The encryption algorithm SEnc takes a k bit key K and plaintext $M \in \{0,1\}^*$ to return a ciphertext C. The decryption algorithm SDec takes K and C to return either a plaintext M or the symbol $\perp$. We require

$$\Pr\left[K \xleftarrow{\$} \{0,1\}^k \ : \ \mathsf{SDec}(K, \mathsf{SEnc}(K, M)) = M \right] = 1$$

for all $M \in \{0,1\}^*$. We also require standard IND-CPA security except that it need only be one-time. This is captured by letting

$$\mathbf{Adv}^{\text{ind-cpa}}_{\mathsf{SE}}(A) \ = \ 2 \cdot \Pr\left[\mathsf{INDCPA}^{A}_{\mathsf{SE}} \Rightarrow \mathsf{true} \right] - 1 \ ,$$

where game $\mathsf{INDCPA}_{\mathsf{SE}}$ is in Figure 4 and A is required to make only one LR query (this is how the one-time requirement is captured), consisting of a pair of equal-length messages.

Construction. Let $\mathsf{KUS} = (\mathsf{KG}, \mathsf{HKU}, \mathsf{UKU}, \mathsf{Sig}, \mathsf{Ver})$ be a key-updating signature scheme. We transform it into a public-channel key-updating signature scheme $\mathsf{PCKUS} = (\mathsf{KUS}, \mathsf{CKG}, (\mathsf{U}, \mathsf{H}))$, where CKG always returns $(\varepsilon, \varepsilon)$, by defining the channel-key-update protocol (U, H) in terms of any secret-key-exchange protocol (I, J) and symmetric encryption scheme $\mathsf{SE} = (\mathsf{SEnc}, \mathsf{SDec})$, both with the same key length k, as follows. The parties first run the secret-key-exchange protocol, with U playing the role of I and H playing the role of J, to agree on a common key K. The helper then encrypts hsk_i under K using SEnc to obtain a ciphertext C which it sends to the user. The latter decrypts C under K using SDec to obtain hsk_i.

We clarify that this particular channel-update protocol is chosen for illustrative purposes. Many others are possible, as the reader will probably see. However, it does include several different instantiations, arising from the different available choices of SKE protocols mentioned above.

Security of our construction. We prove that if the given key-updating signature scheme is KI in the secure-channel model and the secret-key-exchange protocol as well as the symmetric encryption scheme are secure, then the public-channel key-updating signature scheme obtained using our construction is KI under passive attack in the public-channel model.

Theorem 6.1 Let $\mathsf{KUS} = (\mathsf{KG}, \mathsf{HKU}, \mathsf{UKU}, \mathsf{Sig}, \mathsf{Ver})$ be a key-updating signature scheme. Let $\mathsf{PCKUS} = ((\mathsf{KG}, \mathsf{HKU}, \mathsf{UKU}, \mathsf{Sig}, \mathsf{Ver}), \mathsf{CKG}, (\mathsf{U}, \mathsf{H}))$ be the public-channel key-updating signature scheme constructed from KUS, secret-key-exchange protocol (I, J) and symmetric encryption scheme $\mathsf{SE} = (\mathsf{SEnc}, \mathsf{SDec})$ as described above. Let t_{KG}, t_{HKU}, t_{UKU}, t_{Sig}, t_{Ver}, and t_{CKG} denote the running times of the corresponding algorithms, and $t_{(\mathsf{U},\mathsf{H})}$ denote the running time of protocol (U, H). Let A be an adversary against PCKUS, making q queries to oracles **Conv** and **Next**, q_E queries to **Expose**, and q_S queries to **Sign**. Then there exist adversaries E, B, S such that

$$\mathbf{Adv}^{\text{pcki-pa}}_{\mathsf{PCKUS}}(A) \;\le\; q \cdot \mathbf{Adv}^{\text{ske}}_{(\mathsf{I},\mathsf{J})}(E) + q \cdot \mathbf{Adv}^{\text{ind-cpa}}_{\mathsf{SE}}(B) + q \cdot \mathbf{Adv}^{\text{ki}}_{\mathsf{KUS}}(S)\ . \tag{1}$$

Furthermore, the running times of E, B are both $t_{\mathsf{KG}} + t_{\mathsf{CKG}} + q_N \cdot t_{\mathsf{HKU}} + q_N \cdot t_{\mathsf{UKU}} + q_S \cdot t_{\mathsf{Sig}} + q \cdot t_{(\mathsf{U},\mathsf{H})}$, and the running time of S is $t_{\mathsf{CKG}} + \mathcal{O}(q + q_S + q_E) + q \cdot t_{(\mathsf{U},\mathsf{H})}$. Also S makes $q - 1$ queries to its **Expose** oracle and q queries to its **Next** oracle.

Proof overview. The proof that our construction achieves public-channel KI security under passive attack seems easy at first, but there are subtle difficulties arising from the fact that our model allows the adversary to obtain the user coins from the channel-update protocol execution in any stage in which it breaks in. This means that the adversary obtains the session key, and can check whether the ciphertext transmitted by the helper decrypts to the helper secret key for the stages in question, a value it also has from its break-in. Of course, in the real protocol, this will always be true. But the natural simulation is to consider a protocol in which, rather than encrypting the helper key under the session key yielded by the session-key exchange protocol, the helper encrypts a constant under a new, random key. The security of the session-key exchange protocol and the encryption scheme should imply that this makes no difference. However, the adversary can in fact detect the difference between the simulation and the real game because, as we said above, it can obtain the real session key and decrypt the ciphertext under it. To get around this, we guess a stage in which the adversary does not break in, and switch to the simulated key and message only in this stage, using the real key and real message in other stages. But to do this, our simulation needs to know the real message, which is the helper secret key, and the only way to get this is to break in. Luckily, it can do so by consequence of the assumed security of the underlying secure-channel KI scheme, but the result is a discrepancy in resources: even if the adversary against the public channel protocol does very few break-ins, the adversary against the secure-channel protocol breaks-in to $N - 1$ out of N stages. Therefore, it is required that the given secure-channel scheme be secure against $N - 1$ break-ins. Luckily, we have such schemes. For signatures, the schemes of [16] have the desired property. For encryption, some of the original schemes of [15] are threshold and don't have the property, but the scheme of [4] does.

Proof: We use games G_0, G_1, G_2, G_3, G_4 defined in Figure 5. We assume, without loss of generality, that A never asks an oracle query twice and its output $(j, (M, \sigma))$ always satisfies $1 \le j \le q$. We assume that A always makes exactly (as opposed to at most) q **Conv** oracle queries. Game G_0 is simply game $\mathsf{PCKI\text{-}pa}$ of Figure 2 for the case where the channel-key-update protocol is the one we have defined, and so we have

$$\mathbf{Adv}^{\text{pcki-pa}}_{\mathsf{PCKUS}}(A) \;=\; \Pr\left[\, \mathrm{G}_0^A \Rightarrow \mathsf{true} \,\right]\ . \tag{2}$$

Games G_0 and G_1 are identical except for the boxed code in **Finalize**, and on the other hand G_0 does not use g anywhere and thus the events $\mathrm{G}_0^A \Rightarrow \mathsf{true}$ and $g = j$ are independent and the probability of the latter is $1/q$. Hence,

$$\Pr\left[\, \mathrm{G}_1^A \Rightarrow \mathsf{true} \,\right] \;\;=\;\; \Pr\left[\, \mathrm{G}_0^A \Rightarrow \mathsf{true} \wedge g = j \,\right]$$

procedure Initialize
$(pk, usk_0, hsk) \xleftarrow{\$} \mathsf{KG}$
$(uck_0, hck_0) \xleftarrow{\$} \mathsf{CKG}$
$hsk_1 \leftarrow \mathsf{HKU}(1, pk, hsk)$
$\omega_0^U \xleftarrow{\$} \mathsf{COINS}$; $\omega_0^H \xleftarrow{\$} \mathsf{COINS}$
$St_1^U \leftarrow (usk_0, uck_0, \omega_0^U)$
$St_1^H \leftarrow (hsk_1, hck_0, \omega_0^H)$
$a \leftarrow 0$; $S \leftarrow \emptyset$; $E \leftarrow \emptyset$
$g \xleftarrow{\$} \{1, \ldots, q\}$
Return pk

procedure Next()
$a \leftarrow a + 1$
If ($d_a^U = \mathsf{acc}$) then
  $(hsk_a, uck_a) \leftarrow St_a^U$
  $usk_a \leftarrow \mathsf{UKU}(a, pk, hsk_a, usk_{a-1})$
  $\omega_a^U \xleftarrow{\$} \mathsf{COINS}$
  $St_{a+1}^U \leftarrow (usk_a, uck_a, \omega_a^U)$
If ($d_a^H = \mathsf{acc}$) then
  $hck_a \leftarrow St_a^H$
  $hsk_{a+1} \leftarrow \mathsf{HKU}(a + 1, pk, hsk)$
  $\omega_a^H \xleftarrow{\$} \mathsf{COINS}$
  $St_{a+1}^H \leftarrow (hsk_{a+1}, hck_a, \omega_a^H)$

procedure Sign(M)
$(a, \sigma) \xleftarrow{\$} \mathsf{Sig}(a, usk_a, M)$
$S \leftarrow S \cup \{(a, M)\}$
Return (a, σ)

procedure Expose() // G_0, G_1, $\boxed{G_2}$, $\boxed{G_3}$, $\boxed{G_4}$
$E \leftarrow E \cup \{a\}$
$x \leftarrow (usk_a, hsk_a, \varepsilon, \omega_a^U)$
If ($a = g$) then $\boxed{x \leftarrow \perp}$
Return x

procedure Conv() // G_0, G_1, G_2
If ($d_{a+1}^I = \perp \wedge d_{a+1}^J = \perp$) then
  Parse ω_a^H as $\omega_a^J || r$
  $(\text{Conv}, K_1, d_{a+1}^I, K_1, d_{a+1}^J) \leftarrow \mathbf{Run}(\mathsf{I}, \varepsilon, \mathsf{J}, \varepsilon, ; \omega_a^U, \omega_a^J)$
  $C \leftarrow \mathsf{SEnc}(K_1, hsk_{a+1}; r)$
  $St_{a+1}^U \leftarrow (hsk_{a+1}, \varepsilon)$; $St_{a+1}^H \leftarrow \varepsilon$
  Return $(\text{Conv} \parallel C, d_{a+1}^I, d_{a+1}^J)$

procedure Conv() // G_3
If ($d_{a+1}^I = \perp \wedge d_{a+1}^J = \perp$) then
  Parse ω_a^H as $\omega_a^J || r$
  $(\text{Conv}, K_1, d_{a+1}^I, K_1, d_{a+1}^J) \leftarrow \mathbf{Run}(\mathsf{I}, \varepsilon, \mathsf{J}, \varepsilon, ; \omega_a^U, \omega_a^J)$
  $C \leftarrow \mathsf{SEnc}(K_1, hsk_{a+1}; r)$
  If ($a + 1 = g$) then
    $K_0 \xleftarrow{\$} \{0,1\}^k$; $C \xleftarrow{\$} \mathsf{SEnc}(K_0, hsk_{a+1})$
  $St_{a+1}^U \leftarrow (hsk_{a+1}, \varepsilon)$; $St_{a+1}^H \leftarrow \varepsilon$
  Return $(\text{Conv} \parallel C, d_{a+1}^I, d_{a+1}^J)$

procedure Conv() // G_4
If ($d_{a+1}^I = \perp \wedge d_{a+1}^J = \perp$) then
  Parse ω_a^H as $\omega_a^J || r$
  $(\text{Conv}, K_1, d_{a+1}^I, K_1, d_{a+1}^J) \leftarrow \mathbf{Run}(\mathsf{I}, \varepsilon, \mathsf{J}, \varepsilon, ; \omega_a^U, \omega_a^J)$
  $C \leftarrow \mathsf{SEnc}(K_1, hsk_{a+1}; r)$
  If ($a + 1 = g$) then
    $K_0 \xleftarrow{\$} \{0,1\}^k$; $C \xleftarrow{\$} \mathsf{SEnc}(K_0, 0^{|hsk_{a+1}|})$
  $St_{a+1}^U \leftarrow (hsk_{a+1}, \varepsilon)$; $St_{a+1}^H \leftarrow \varepsilon$
  Return $(\text{Conv} \parallel C, d_{a+1}^I, d_{a+1}^J)$

procedure Finalize$(M, (j, \sigma))$ // G_0, $\boxed{G_1}$, $\boxed{G_2}$, $\boxed{G_3}$, $\boxed{G_4}$
Return ($\boxed{g = j \wedge}$ $j \notin E \wedge (j, M) \notin S \wedge \mathsf{Ver}(pk, M, (j, \sigma)) = 1$)

Figure 5: Games $\mathrm{G}_0, \mathrm{G}_1, \mathrm{G}_2, \mathrm{G}_3, \mathrm{G}_4$ used for the proof of Theorem 6.1. The boxed code in **Finalize** is omitted in G_0 but present for the other games. The boxed code in **Expose** is omitted in G_0, G_1 but present for the other games.

$$= \Pr\left[\mathrm{G}_0^A \Rightarrow \mathsf{true}\right] \cdot \Pr\left[g = j\right]$$
$$= \Pr\left[\mathrm{G}_0^A \Rightarrow \mathsf{true}\right] \cdot \frac{1}{q} \quad . \tag{3}$$

The difference between G_2 and G_1 is that the former includes the boxed code in **Expose**. However, any execution of G_2 with A in which the outcome is true must have $g = j$ and $j \notin E$, so the boxed

code would not have been executed. This means that

$$\Pr\left[\,\mathrm{G}_2^A \Rightarrow \mathsf{true}\,\right] = \Pr\left[\,\mathrm{G}_1^A \Rightarrow \mathsf{true}\,\right]. \tag{4}$$

From equations (2), (3) and (4), we have

$$\mathbf{Adv}^{\text{pcki-pa}}_{\mathsf{PCKUS}}(A) \;=\; q \cdot \Pr\left[\,\mathrm{G}_1^A \Rightarrow \mathsf{true}\,\right] \;=\; q \cdot \Pr\left[\,\mathrm{G}_2^A \Rightarrow \mathsf{true}\,\right]. \tag{5}$$

We will build E, B, S so that

$$\Pr\left[\,\mathrm{G}_2^A \Rightarrow \mathsf{true}\,\right] - \Pr\left[\,\mathrm{G}_3^A \Rightarrow \mathsf{true}\,\right] \;\leq\; \mathbf{Adv}^{\text{ske}}_{(\mathsf{I},\mathsf{J})}(E) \tag{6}$$

$$\Pr\left[\,\mathrm{G}_3^A \Rightarrow \mathsf{true}\,\right] - \Pr\left[\,\mathrm{G}_4^A \Rightarrow \mathsf{true}\,\right] \;\leq\; \mathbf{Adv}^{\text{ind-cpa}}_{\mathsf{SE}}(B) \tag{7}$$

$$\Pr\left[\,\mathrm{G}_4^A \Rightarrow \mathsf{true}\,\right] \;\leq\; \mathbf{Adv}^{\text{ki}}_{\mathsf{KUS}}(S)\,. \tag{8}$$

Assuming this for now, we show how to conclude. By adding equations (6), (7) and (8), we have

$$\Pr\left[\,\mathrm{G}_2^A \Rightarrow \mathsf{true}\,\right] \;\leq\; \mathbf{Adv}^{\text{ske}}_{(\mathsf{I},\mathsf{J})}(E) + \mathbf{Adv}^{\text{ind-cpa}}_{\mathsf{SE}}(B) + \mathbf{Adv}^{\text{ki}}_{\mathsf{KUS}}(S)\,. \tag{9}$$

From equations (9) and (5), we have equation (1).

We now show how to build adversary E against game $\mathsf{SKE}_{(\mathsf{I},\mathsf{J})}$. Adversary E begins by executing the code of the **Initialize** procedure of G_2, thereby defining for itself all the variables there. It then starts running A on input pk, which is one of the variables it just defined. It answers A's queries to its **Next**, **Expose** and **Sign** oracles exactly as G_2 does, and answers queries to the **Conv** oracle via the following procedure:

procedure **Conv**()
If $(\, d^I_{a+1} = \perp \wedge d^J_{a+1} = \perp\,)$ then
  Parse ω_a^H as $\omega_a^J || r$
  $(\text{Conv}, K_1, d^I_{a+1}, K_1, d^J_{a+1}) \leftarrow \mathbf{Run}(\mathsf{I}, \varepsilon, \mathsf{J}, \varepsilon; \omega_a^U, \omega_a^J)$; $C \leftarrow \mathsf{SEnc}(K_1, hsk_{a+1}; r)$
  If $(\, a+1 = g\,)$ then $(\text{Conv}, d^I_{a+1}, d^J_{a+1}, K_b) \xleftarrow{\$} \mathbf{Conv}()$; $C \xleftarrow{\$} \mathsf{SEnc}(K_b, hsk_{a+1})$
  Return $(\text{Conv} \parallel C, d^I_{a+1}, d^J_{a+1})$

In the 4th line of the code above, E invokes its own **Conv** oracle. Finally, A outputs $(j, (M, \sigma))$. Adversary E outputs 1 if $g = j \wedge j \notin E \wedge (j, M) \notin S \wedge \mathsf{Ver}(pk, M, (j, \sigma)) = 1$, and otherwise it outputs 0. We have

$$\Pr\left[\,\mathsf{SKE}^E_{\mathsf{I},\mathsf{J}} \Rightarrow \mathsf{true} \mid\; b = 1\,\right] \;=\; \Pr\left[\,\mathrm{G}_2^A \Rightarrow \mathsf{true}\,\right]$$

$$\Pr\left[\,\mathsf{SKE}^E_{\mathsf{I},\mathsf{J}} \Rightarrow \mathsf{true} \mid\; b = 0\,\right] \;=\; \Pr\left[\,\mathrm{G}_3^A \Rightarrow \mathsf{true}\,\right]$$

Subtracting, we get equation (6).

Next, we show how to build adversary B against game $\mathsf{INDCPA}_{\mathsf{SE}}$. Adversary B begins by executing the code of the **Initialize** procedure of G_3, thereby defining for itself all the variables there. It then starts running A on input pk. It answers A's queries to its **Next**, **Expose** and **Sign** oracles exactly as G_3 does, and answers queries to the **Conv** oracle via the following procedure:

procedure **Conv**()
If $(\, d^I_{a+1} = \perp \wedge d^J_{a+1} = \perp\,)$ then

Parse ω_a^H as $\omega_a^J||r$
$(\text{Conv}, K_1, d_{a+1}^I, K_1, d_{a+1}^J) \leftarrow \mathbf{Run}(\mathsf{I}, \varepsilon, \mathsf{J}, \varepsilon; \omega_a^U, \omega_a^J)$; $C \leftarrow \mathsf{SEnc}(K_1, hsk_{a+1}; r)$
If ($a + 1 = g$) then $C \xleftarrow{\$} \mathbf{LR}(hsk_{a+1}, 0^{|hsk_{a+1}|})$
Return $(\text{Conv} \parallel C, d_{a+1}^I, d_{a+1}^J)$

In the 4th line of the code above, B queries its **LR** oracle. Finally, A outputs $(j, (M, \sigma))$. Adversary B outputs 1 if $g = j \wedge j \notin E \wedge (j, M) \notin S \wedge \mathsf{Ver}(pk, M, (j, \sigma)) = 1$, and otherwise it outputs 0. We have

$$\begin{aligned} \Pr\left[\, \mathsf{INDCPA}_{\mathsf{SE}}^B \Rightarrow \mathsf{true} \mid b = 0 \,\right] &= \Pr\left[\, \mathrm{G}_3^A \Rightarrow \mathsf{true} \,\right] \\ \Pr\left[\, \mathsf{INDCPA}_{\mathsf{SE}}^B \Rightarrow \mathsf{true} \mid b = 1 \,\right] &= \Pr\left[\, \mathrm{G}_4^A \Rightarrow \mathsf{true} \,\right] \end{aligned}$$

Subtracting, we get equation (7).

Finally, we show how adversary S works against game KIS. S is given input pk. It selects $\omega_0^U, \omega_0^H \in \mathsf{COINS}$ and $g \in \{1, \ldots, q\}$ independently at random, sets $a = 0$, and starts running A on input pk. S answers A's queries to the **Conv** oracle via the following procedure:

procedure **Conv**()
If ($d_{a+1}^I = \perp \wedge d_{a+1}^J = \perp$) then
Parse ω_a^H as $\omega_a^J||r$
$(\text{Conv}, K_1, d_{a+1}^I, K_1, d_{a+1}^J) \leftarrow \mathbf{Run}(\mathsf{I}, \varepsilon, \mathsf{J}, \varepsilon; \omega_a^U, \omega_a^J)$
Next()
If ($a + 1 = g$) then $K_0 \xleftarrow{\$} \{0,1\}^k$; $C \xleftarrow{\$} \mathsf{SEnc}(K_0, 0^{|hsk_{a+1}|})$
else $(usk_{a+1}, hsk_{a+1}) \leftarrow \mathbf{Expose}()$; $C \leftarrow \mathsf{SEnc}(K_1, hsk_{a+1}; r)$
Return $(\text{Conv} \parallel C, d_{a+1}^I, d_{a+1}^J)$

In the 4th line of the code above, S queries its **Next** oracle, and in the 6th line, it queries its **Expose** oracle. S answers A's queries to the **Next** oracle by incrementing a and selecting $\omega_a^U, \omega_a^H \in \mathsf{COINS}$ independently at random. It answers A's queries to its **Expose** and **Sign** oracles via its own corresponding oracles (setting $uck_a = \varepsilon$). Finally, A outputs $(j, (M, \sigma))$ and S returns this same output. From the above, we know that if A wins game G_4, the event of $g = j$ must happen. In addition, S never queries its own **Expose** oracle in stage g. Thus S can output the same signature $(g, (M, \sigma))$ as A and win the game KIS. So we have equation (8). ∎

7 Intrusion resilience in the secure-channel model

In an intrusion-resilient signature scheme both the user's and the helper's secret keys evolve with time. In addition, these schemes include a refresh procedure for helper and user such that if a refresh is run between the compromise of the user or helper and the compromise of the other, then the system remains secure, except for the current stage. In our security definition below, we denote the number of refreshes per stage by RN. Our definition differs from that of [20] in some details, but this does not affect the results.

A *signer-base key-updating signature scheme* $\mathsf{SBKUS} = (\mathsf{KG}, \mathsf{HKU}, \mathsf{UKU}, \mathsf{HKR}, \mathsf{UKR}, \mathsf{Sig}, \mathsf{Ver})$ is specified by seven algorithms with the following functionality. The randomized *key-generation algorithm* KG returns $(pk, usk_{0.0}, hsk_{0.0})$, where pk is the user public key, $usk_{0.0}$ is the *stage 0 user secret key*, and $hsk_{0.0}$ is the *stage 0 helper key*. The user is initialized with $pk, usk_{0.0}$, while the

procedure Initialize
$(pk, usk_{0.0}, hsk_{0.0}) \xleftarrow{\$} \mathsf{KG}$
$a \leftarrow 0\,;\ r \leftarrow 0\,;\ S \leftarrow \emptyset$
Return pk

procedure Next()
$a \leftarrow a + 1$
$(hsk_{a.0}, hsku_a) \xleftarrow{\$} \mathsf{HKU}(a, pk, hsk_{(a-1).r})$
$usk_{a.0} \xleftarrow{\$} \mathsf{UKU}(a, pk, hsku_a, usk_{(a-1).r})$
$(hsk_{a.1}, hskr_{a.1}) \xleftarrow{\$} \mathsf{HKR}(a, pk, hsk_{a.0})$
$usk_{a.1} \xleftarrow{\$} \mathsf{UKR}(a, pk, hskr_{a.1}, usk_{a.0})$
$r \leftarrow 1$

procedure Finalize$(M, (j, \sigma))$
If $((j, M) \in S \vee \mathsf{Ver}(pk, M, (j, \sigma)) = 0)$ then
  Return false
If $\exists s(1 \le s \le RN \wedge ExpU_{j.s})$ then Return false
If $\exists i \exists s(1 \le i < j \wedge 1 \le s \le RN \wedge$
  $ExpU_{i.s} \wedge ExpH_{i.s})$ then Return false
Return true

procedure Refresh()
$r \leftarrow r + 1$
$(hsk_{a.r}, hskr_{a.r}) \xleftarrow{\$} \mathsf{HKR}(a, pk, hsk_{a.(r-1)})$
$usk_{a.r} \xleftarrow{\$} \mathsf{UKR}(a, pk, hskr_{a.r}, usk_{a.(r-1)})$

procedure Expose(Type)
If (Type = "U") then
  $ExpU_{a.r} \leftarrow$ true ; Return $usk_{a.r}$
If (Type = "H") then
  $ExpH_{a.r} \leftarrow$ true ; Return $hsk_{a.r}$
If (Type = "P") then
  If $(ExpU_{(a-1).RN})$ then
    $ExpU_{a.1} \leftarrow$ true ; Return $(hsku_a, hskr_{a.1})$
If (Type = "R") then
  If $(ExpU_{a.(r-1)})$ then
    $ExpU_{a.r} \leftarrow$ true ; Return $hskr_{a.r}$
Return $\perp$

procedure Sign(M)
$(a, \sigma) \xleftarrow{\$} \mathsf{Sig}(a, usk_{a.r}, M)$
$S \leftarrow S \cup \{(a, M)\}$
Return (a, σ)

Figure 6: Game IRS used to define IR signatures in the secure-channel model.

helper is initialized with $pk, hsk_{0.0}$. At the start of stage $a \ge 1$, if r refreshes have been made since the last update, where $0 \le r \le RN$, the helper applies the randomized *helper key-update algorithm* HKU to $a, pk, hsk_{(a-1).r}$ to obtain a *stage a helper key* $hsk_{a.0}$ and a *stage a helper update key* $hsku_a$. The helper discards (erases) $hsk_{(a-1).r}$. Then it applies the randomized *helper key-refresh algorithm* HKR to $a, pk, hsk_{a.0}$ to obtain a *stage a helper key* $hsk_{a.1}$ and a *stage a helper refresh key* $hskr_{a.1}$. It discards $hsk_{a.0}$. The helper update key $hsku_a$ and the helper refresh key $hskr_{a.1}$ are then assumed to be conveyed to the user via a secure channel. The user receives $hsku_a$ and $hskr_{a.1}$ from the helper, and applies the randomized *user key-update algorithm* UKU to $a, pk, hsku_a, usk_{(a-1).r}$ to obtain a *stage a user secret key* $usk_{a.0}$. The user then discards $usk_{(a-1).r}$. Then the user applies the randomized *user key-refresh algorithm* UKR to $a, pk, hskr_{a.1}, usk_{a.0}$ to obtain a *stage a user secret key* $usk_{a.1}$. The user then discards $usk_{a.0}$. In stage a, if r refreshes have been made, where $1 \le r < RN$, the helper can apply the helper key-refresh algorithm HKR to $a, pk, hsk_{a.r}$ to obtain a stage a helper key $hsk_{a.(r+1)}$ and a stage a helper refresh key $hskr_{a.(r+1)}$. It then discards $hsk_{a.r}$. The helper refresh key $hskr_{a.(r+1)}$ is assumed to be conveyed to the user via a secure channel. The user receives $hskr_{a.(r+1)}$ from the helper, and applies the user key-refresh algorithm UKR to $a, pk, hskr_{a.(r+1)}, usk_{a.r}$ to obtain a stage a user secret key $usk_{a.(r+1)}$. The user then discards $usk_{a.r}$. In stage a, if r refreshes have been made, where $1 \le r \le RN$, the user can apply the randomized *signing algorithm* Sig to a, stage a secret key $usk_{a.r}$, and a message $M \in \{0,1\}^*$ to obtain a pair (a, σ), consisting of the stage number a and a signature σ. During stage a anyone can apply the deterministic *verification algorithm* Ver to pk, a message M, and a pair (i, σ) to obtain either 1, indicating acceptance, or 0, indicating rejection. We require that if (i, σ), where $1 \le i \le a$, was produced by applying the signing algorithm to $i, usk_{i.r}, M$ then $\mathsf{Ver}(pk, M, (i, \sigma)) = 1$.

SECURITY. Consider game IRS of Figure 6. The **Initialize** procedure provides adversary A with input pk. A can call its **Next** oracle to move the system into the next stage. It can call its **Refresh** oracle to refresh the helper and user's secret keys. It may break in during the current stage by calling its **Expose** oracle. Four types of **Expose** queries are allowed. Query "U" returns the user secret key for the current stage. Query "H" returns the helper key for that stage. Query "P" returns the helper update and helper refresh keys for the current stage. Query "R" returns the helper refresh key for that stage. A may obtain signatures for messages of its choice during the current stage by calling its **Sign** oracle.

For any stage a and any refresh number r, $1 \le r \le RN$, user secret key $usk_{a.r}$ is said to be *exposed* if $ExpU_{a.r} = \mathsf{true}$. Helper key $hsk_{a.r}$ is said to be *exposed* if $ExpH_{a.r} = \mathsf{true}$. Signer-base key-updating signature scheme SBKUS is said to be *a-compromised* if $usk_{a.r}$ is exposed for some r, $1 \le r \le RN$, or there exists $i < a$ such that $usk_{i.r}$ and $hsk_{i.r}$ are exposed for some r, $1 \le r \le RN$.

To win, A must output a message M and a signature (j, σ) such that M was not queried to **Sign** during stage j, $\mathsf{Ver}(pk, M, (j, \sigma)) = 1$, and the scheme is not j-compromised. A's advantage is

$$\mathbf{Adv}^{\mathrm{ir}}_{\mathsf{SBKUS}}(A) \;=\; \Pr\left[\,\mathsf{IRS}^{A} \Rightarrow \mathsf{true}\,\right].$$

We adopt the convention that the *running time* of an adversary A is the execution time of the entire game, including the time taken for initialization, the time taken by the oracles to compute replies to the adversary's queries, and the time taken for finalization.

8 Intrusion resilience in the public-channel model

PUBLIC-CHANNEL SIGNER-BASE KEY UPDATING SIGNATURE SCHEMES. A *public-channel signer-base key-updating signature scheme* is a tuple $\mathsf{PCSBKUS} = (\mathsf{SBKUS}, \mathsf{CKG}, (\mathsf{UU}, \mathsf{HU}), (\mathsf{UR}, \mathsf{HR}))$, where $\mathsf{SBKUS} = (\mathsf{KG}, \mathsf{HKU}, \mathsf{UKU}, \mathsf{HKR}, \mathsf{UKR}, \mathsf{Sig}, \mathsf{Ver})$ is a signer-base key-updating signature scheme, CKG is the channel-key-generation algorithm, and the channel-key-update protocol $(\mathsf{UU}, \mathsf{HU})$ and channel-key-refresh protocol $(\mathsf{UR}, \mathsf{HR})$ are each pairs of interactive algorithms to be run by user and helper, respectively. Let RN denote the number of refreshes per stage in SBKUS. We now explain how the system runs.

Algorithm CKG returns the *stage 0 user channel key* $uck_{0.0}$ and the *stage 0 helper channel key* $hck_{0.0}$. When the user is initialized, in addition to the public key pk and stage 0 user secret key $usk_{0.0}$ produced by KG, the user is given $uck_{0.0}$. When the helper is initialized, in addition to pk and the stage 0 helper key $hsk_{0.0}$ (also generated by KG), the helper is given $hck_{0.0}$.

In any stage $a \ge 0$, if r refreshes have been made since the last update, where $0 \le r \le RN$, the user holds a stage a user secret key $usk_{a.r}$ and a stage a user channel key $uck_{a.r}$. The helper holds a stage a helper key $hsk_{a.r}$ and a stage a helper channel key $hck_{a.r}$. At the start of stage $a+1$, the helper computes $(hsk_{(a+1).0}, hsku_{a+1}) \xleftarrow{\$} \mathsf{HKU}(a+1, pk, hsk_{a.r})$, discards (erases) $hsk_{a.r}$, computes $(hsk_{(a+1).1}, hskr_{(a+1).1}) \xleftarrow{\$} \mathsf{HKR}(a+1, pk, hsk_{(a+1).0})$, and discards $hsk_{(a+1).0}$. Then the parties engage in the channel-key-update protocol $(\mathsf{UU}, \mathsf{HU})$. The local input of UU is the stage a user secret key $usk_{a.r}$, the stage a user channel key $uck_{a.r}$, and some random coins $\omega^U_{a.r}$; while the local input of HU is the stage $a+1$ helper update key $hsku_{a+1}$, the stage $a+1$ helper refresh key $hskr_{(a+1).1}$, the stage a helper channel key $hck_{a.r}$, and some random coins $\omega^H_{a.r}$. After the interaction, the expected local output of UU is $hsku_{a+1}$, $hskr_{(a+1).1}$, plus the stage $a+1$ user channel key $uck_{(a+1).1}$; while the expected local output of HU is the stage $a+1$ helper channel key $hck_{(a+1).1}$. Once the protocol has completed, the user immediately updates its secret key by

computing $usk_{(a+1).0} \xleftarrow{\$} \mathsf{UKU}(a+1, pk, hsku_{a+1}, usk_{a.r})$, discarding $usk_{a.r}$, computing $usk_{(a+1).1} \xleftarrow{\$} \mathsf{UKR}(a+1, pk, hskr_{(a+1).1}, usk_{(a+1).0})$, and discarding $usk_{(a+1).0}$. The user also discards its previous channel key $uck_{a.r}$. The helper discards its previous channel key $hck_{a.r}$ as well. We require the natural correctness condition, namely that the stage $a+1$ helper update key and the stage $a+1$ helper refresh key produced by UU in the interaction in which UU has input $usk_{a.r}, uck_{a.r}, \omega^U_{a.r}$ and HU has input $hsku_{a+1}, hskr_{(a+1).1}, hck_{a.r}, \omega^H_{a.r}$, are, respectively, $hsku_{a+1}$ and $hskr_{(a+1).1}$ with probability one. In addition, we require that at the end of the interaction, UU's decision $d^U_{(a+1).1}$ and HU's decision $d^H_{(a+1).1}$ are both acc.

In stage a, if r refreshes have been made, where $1 \le r \le RN$, the helper can compute $(hsk_{a.(r+1)}, hskr_{a.(r+1)}) \xleftarrow{\$} \mathsf{HKR}(a, pk, hsk_{a.r})$ and discard $hsk_{a.r}$. Then the parties engage in the channel-key-refresh protocol $(\mathsf{UR}, \mathsf{HR})$. The local input of UR is the stage a user secret key $usk_{a.r}$, the stage a user channel key $uck_{a.r}$, and some random coins $\omega^U_{a.r}$; while the local input of HR is the stage a helper refresh key $hskr_{a.(r+1)}$, the stage a helper channel key $hck_{a.r}$, and some random coins $\omega^H_{a.r}$. After the interaction, the expected local output of UR is $hskr_{a.(r+1)}$ plus the stage a user channel key $uck_{a.(r+1)}$; while the expected local output of HR is the stage a helper channel key $hck_{a.(r+1)}$. Once the protocol has completed, the user updates its secret key by computing $usk_{a.(r+1)} \xleftarrow{\$} \mathsf{UKR}(a, pk, hskr_{a.(r+1)}, usk_{a.r})$, and discarding $usk_{a.r}$. The user also discards its previous channel key $uck_{a.r}$. The helper discards its previous channel key $hck_{a.r}$ as well. We require the natural correctness condition, namely that the stage $a+1$ helper refresh key produced by UR in the interaction in which UR has input $usk_{a.r}, uck_{a.r}, \omega^U_{a.r}$ and HR has input $hskr_{a.(r+1)}, hck_{a.r}, \omega^H_{a.r}$, is $hskr_{a.(r+1)}$ with probability one. In addition, we require that at the end of the interaction, UR's decision $d^U_{a.(r+1)}$ and HU's decision $d^H_{a.(r+1)}$ are both acc. SECURITY. We proceed to formalize a notion of security for public-channel signer-base key-updating signature schemes: intrusion resilience under passive attacks. We first provide a definition and then explanations. Let $\mathsf{PCSBKUS} = ((\mathsf{KG}, \mathsf{HKU}, \mathsf{UKU}, \mathsf{HKR}, \mathsf{UKR}, \mathsf{Sig}, \mathsf{Ver}), \mathsf{CKG}, (\mathsf{UU}, \mathsf{HU}), (\mathsf{UR}, \mathsf{HR}))$ be a public-channel signer-base key-updating signature scheme. We consider an adversary A interacting with the game of Figure 7.

The **Initialize** procedure gives A input pk. A is provided with oracles **Next**, **Refresh**, **Expose**, **UConv**, **RConv**, and **Sign**. It may query the oracles adaptively, in any order it wants, with the following restrictions: every query to oracle **UConv** is immediately followed by a query to oracle **Next**, every query to oracle **RConv** is immediately followed by a query to oracle **Refresh**, and in every stage the adversary makes exactly RN **Refresh** queries. Eventually, A outputs a message M and a signature (j, σ) and halts. A passive adversary is said to win if game $\mathsf{PCIR\text{-}pa}$ returns true, meaning M was not queried to **Sign** during stage j, $\mathsf{Ver}(pk, M, (j, \sigma)) = 1$, no **Expose** queries of type "U" were made during stage j, and during all previous stages i, $1 \le i < j$, no **Expose** queries of type "U" and type "H" were made without a refresh in between them. A's pa-advantage is

$$\mathbf{Adv}^{\text{pcir-pa}}_{\mathsf{PCSBKUS}}(A) \;=\; \Pr\left[\,\mathsf{PCIR\text{-}pa}^A \Rightarrow \mathsf{true}\,\right].$$

Again, we adopt the convention that the *running time* of an adversary A is the execution time of the entire game, including the time taken for initialization, the time taken by the oracles to compute replies to the adversary's queries, and the time taken for finalization.

9 Possibility of public-channel IR under passive attack

Given a signer-base key-updating signature scheme in the secure-channel model, we show in this section how to transform it into a signer-base key-updating signature scheme secure against passive attack in the public-channel model, by using secret-key-exchange protocols and symmetric

Figure 7: Game PCIR-pa used to define public-channel intrusion resilience under passive attack.

encryption schemes.

Construction. Let $\mathsf{SBKUS} = (\mathsf{KG}, \mathsf{HKU}, \mathsf{UKU}, \mathsf{HKR}, \mathsf{UKR}, \mathsf{Sig}, \mathsf{Ver})$ be a signer-base key-updating signature scheme. We transform it into a public-channel signer-base key-updating signature scheme $\mathsf{PCSBKUS} = (\mathsf{SBKUS}, \mathsf{CKG}, (\mathsf{UU}, \mathsf{HU}), (\mathsf{UR}, \mathsf{HR}))$, where CKG always returns $(\varepsilon, \varepsilon)$, by defining both the channel-key-update protocol $(\mathsf{UU}, \mathsf{HU})$ and the channel-key-refresh protocol $(\mathsf{UR}, \mathsf{HR})$ in terms

of any secret-key-exchange protocol (I, J) and symmetric encryption scheme $\mathsf{SE} = (\mathsf{SEnc}, \mathsf{SDec})$, the protocol and the encryption with the same key length k, as follows. During each update (resp., refresh), the parties first run the secret-key-exchange protocol, with U playing the role of I and H playing the role of J, to agree on a common key K. The helper then encrypts $(hsku_i, hskr_{i.1})$ (resp., $hskr_{i.r}$) under K using SEnc to obtain a ciphertext C which it sends to the user. The latter decrypts C under K using SDec to obtain $(hsku_i, hskr_{i.1})$ (resp., $hskr_{i.r}$).

Security of our construction. We prove that if the given signer-base key-updating signature scheme is intrusion-resilience in the secure-channel model and the secret-key-exchange protocol as well as the symmetric encryption scheme are secure, then the public-channel signer-base key-updating signature scheme obtained using our construction is IR under passive attack in the public-channel model.

Theorem 9.1 Let $\mathsf{SBKUS} = (\mathsf{KG}, \mathsf{HKU}, \mathsf{UKU}, \mathsf{HKR}, \mathsf{UKR}, \mathsf{Sig}, \mathsf{Ver})$ be a signer-base key-updating signature scheme. Let $\mathsf{PCSBKUS} = (\mathsf{SBKUS}, \mathsf{CKG}, (\mathsf{UU}, \mathsf{HU}), (\mathsf{UR}, \mathsf{HR}))$ be the public-channel signer-base key-updating signature scheme constructed from SBKUS, secret-key-exchange protocol (I, J) and symmetric encryption scheme $\mathsf{SE} = (\mathsf{SEnc}, \mathsf{SDec})$ as described above. Let t_{KG}, t_{HKU}, t_{UKU}, t_{HKR}, t_{UKR}, t_{Sig}, t_{Ver}, and t_{CKG} denote the running times of the corresponding algorithms, and $t_{(\mathsf{UU},\mathsf{HU})}$, $t_{(\mathsf{UR},\mathsf{HR})}$ denote the running times of protocols $(\mathsf{UU}, \mathsf{HU})$ and $(\mathsf{UR}, \mathsf{HR})$ respectively. Let A be an adversary against $\mathsf{PCSBKUS}$, making q_U queries to oracle **UConv** and **Next**, q_R queries to **RConv** and **Refresh**, q_E queries to **Expose**, q_S queries to **Sign**. Then there exist adversaries E, B, S such that

$$\mathbf{Adv}^{\text{pcir-pa}}_{\mathsf{PCSBKUS}}(A) \;\le\; q_U \cdot \mathbf{Adv}^{\text{ske}}_{(\mathsf{I},\mathsf{J})}(E) + q_U \cdot \mathbf{Adv}^{\text{ind-cpa}}_{\mathsf{SE}}(B) + q_U \cdot \mathbf{Adv}^{\text{ir}}_{\mathsf{SBKUS}}(S)\,. \qquad (10)$$

Furthermore, the running times of E, B are both $t_{\mathsf{KG}} + t_{\mathsf{CKG}} + q_U \cdot (t_{\mathsf{HKU}} + t_{\mathsf{UKU}}) + q_R \cdot (t_{\mathsf{HKR}} + t_{\mathsf{UKR}}) + q_S \cdot t_{\mathsf{Sig}} + q_U \cdot t_{(\mathsf{UU},\mathsf{HU})} + q_R \cdot t_{(\mathsf{UR},\mathsf{HR})}$ and the running time of S is $t_{\mathsf{CKG}} + \mathcal{O}(q_U + q_R + q_S + q_E) + q_U \cdot t_{(\mathsf{UU},\mathsf{HU})} + q_R \cdot t_{(\mathsf{UR},\mathsf{HR})}$. Also S makes $3q_E + q_U + q_R - 5$ queries to its **Expose** oracle, q_U queries to its **Next** oracle and q_R queries to its **Refresh** oracle.

Proof: We use games G_0, G_1, G_2, G_3, G_4 defined in Figure 8. We assume, without loss of generality, that A never asks an oracle query twice and its output $(j, (M, \sigma))$ always satisfies $1 \le j \le q_U$. We assume that A always makes exactly (as opposed to at most) q_U **UConv** oracle queries. Game G_0 is simply the game $\mathsf{PCIR\text{-}pa}$ for the case where the channel-key-update protocol and channel-key-refresh protocol are what we have defined and so we have

$$\mathbf{Adv}^{\text{pcir-pa}}_{\mathsf{PCSBKUS}}(A) \;=\; \Pr\left[\,\mathrm{G}_0^A \Rightarrow \mathsf{true}\,\right]. \qquad (11)$$

Games G_0 and G_1 are identical except for the boxed code in **Finalize**, and on the other hand G_0 does not use g anywhere and thus the events $\mathrm{G}_0^A \Rightarrow \mathsf{true}$ and $g = j$ are independent and the probability of the latter is $1/q_U$. Hence,

$$\begin{aligned}
\Pr\left[\,\mathrm{G}_1^A \Rightarrow \mathsf{true}\,\right] &= \Pr\left[\,\mathrm{G}_0^A \Rightarrow \mathsf{true} \wedge g = j\,\right] \\
&= \Pr\left[\,\mathrm{G}_0^A \Rightarrow \mathsf{true}\,\right] \cdot \Pr\left[\,g = j\,\right] \\
&= \Pr\left[\,\mathrm{G}_0^A \Rightarrow \mathsf{true}\,\right] \cdot \frac{1}{q_U}\;. \qquad (12)
\end{aligned}$$

The difference between G_2 and G_1 is that the former includes the boxed code in **Expose**. However, any execution of G_2 with A in which the outcome is true must have $g = j$, $\forall s 1 \le s \le RN, ExpU_{g.s} \neq$

true and $\forall i \forall s, 1 \le i < g \wedge 1 \le s \le RN, ExpU_{i.s} \ne \mathsf{true} \vee ExpH_{i.s} \ne \mathsf{true}$ (we refer to such a forgery satisfying these requirements later as a successful one). So the boxed code would not have been executed. This means that

$$\Pr\left[\,\mathrm{G}_2^A \Rightarrow \mathsf{true}\,\right] = \Pr\left[\,\mathrm{G}_1^A \Rightarrow \mathsf{true}\,\right] . \tag{13}$$

From equations (11), (12) and (13), we have

$$\mathbf{Adv}^{\text{pcir-pa}}_{\mathsf{PCKUS}}(A) \;=\; q_U \cdot \Pr\left[\,\mathrm{G}_1^A \Rightarrow \mathsf{true}\,\right] \;=\; q_U \cdot \Pr\left[\,\mathrm{G}_2^A \Rightarrow \mathsf{true}\,\right] . \tag{14}$$

We will build E, B, S so that

$$\Pr\left[\,\mathrm{G}_2^A \Rightarrow \mathsf{true}\,\right] - \Pr\left[\,\mathrm{G}_3^A \Rightarrow \mathsf{true}\,\right] \;\le\; \mathbf{Adv}^{\text{ske}}_{(\mathsf{I},\mathsf{J})}(E) \tag{15}$$

$$\Pr\left[\,\mathrm{G}_3^A \Rightarrow \mathsf{true}\,\right] - \Pr\left[\,\mathrm{G}_4^A \Rightarrow \mathsf{true}\,\right] \;\le\; \mathbf{Adv}^{\text{ind-cpa}}_{\mathsf{SE}}(B) \tag{16}$$

$$\Pr\left[\,\mathrm{G}_4^A \Rightarrow \mathsf{true}\,\right] \;\le\; \mathbf{Adv}^{\text{ir}}_{\mathsf{SBKUS}}(S) . \tag{17}$$

Assuming this for now, we show how to conclude. By adding equations (15), (16) and (17), we have

$$\Pr\left[\,\mathrm{G}_2^A \Rightarrow \mathsf{true}\,\right] \;\le\; \mathbf{Adv}^{\text{ske}}_{(\mathsf{I},\mathsf{J})}(E) + \mathbf{Adv}^{\text{ind-cpa}}_{\mathsf{SE}}(B) + \mathbf{Adv}^{\text{ir}}_{\mathsf{SBKUS}}(S) . \tag{18}$$

From equations (18) and (14), we have equation (10).

We now show how to build adversary E against game $\mathsf{SKE}_{(\mathsf{I},\mathsf{J})}$. Adversary E begins by executing the code of the **Initialize** procedure of G_2, thereby defining for itself all the variables there. It then starts running A on input pk, which is one of the variables it just defined. It answers A's queries to its **Next**, **Refresh**, **Expose** and **Sign** oracles exactly as G_2 does, and answers queries to the **UConv** and **RConv** oracles via the following procedures:

procedure **UConv**()
If ($d^I_{(a+1).1} = \perp \wedge d^J_{(a+1).1} = \perp$) then
  Parse $\omega^H_{a.RN}$ as $\omega^J_{a.RN} \parallel r$
  $(\text{Conv}, K_1, d^I_{(a+1).1}, K_1, d^J_{(a+1).1}) \xleftarrow{\$} \mathbf{Run}(\mathsf{I}, \varepsilon, \mathsf{J}, \varepsilon, ; \omega^U_{a.RN}, \omega^J_{a.RN})$
  $C \leftarrow \mathsf{SEnc}(K_1, (hsku_{a+1}, hskr_{(a+1).1}); r)$
  If ($a + 1 = g \wedge ExpU_{a.RN}$) then
    $(\text{Conv}, d^I, d^J, K_b) \xleftarrow{\$} \mathbf{Conv}()$; $C \xleftarrow{\$} \mathsf{SEnc}(K_b, (hsku_{a+1}, hskr_{(a+1).1}))$
Return $(\text{Conv} \parallel C, d^I_{(a+1).1}, d^J_{(a+1).1})$

procedure **RConv**()
If ($d^I_{a.(r+1)} = \perp \wedge d^J_{a.(r+1)} = \perp$) then
  Parse $\omega^H_{a.r}$ as $\omega^J_{a.r} \parallel r$
  $(\text{Conv}, K_1, d^I_{a.(r+1)}, K_1, d^J_{a.(r+1)}) \xleftarrow{\$} \mathbf{Run}(\mathsf{I}, \varepsilon, \mathsf{J}, \varepsilon; \omega^U_{a.r}, \omega^J_{a.r})$
  $C \leftarrow \mathsf{SEnc}(K_1, hskr_{a.(r+1)}; r)$
  If ($a = g \wedge (\exists s, 1 \le s \le RN, ExpU_{a.s})$) then
    $(\text{Conv}, d^I, d^J, K_b) \xleftarrow{\$} \mathbf{Conv}()$; $C \xleftarrow{\$} \mathsf{SEnc}(K_b, hskr_{a.(r+1)})$
Return $(\text{Conv} \parallel C, d^I_{a.(r+1)}, d^J_{a.(r+1)})$

In the 6th line of each procedure above, E invokes its own **Conv** oracle. Finally, A outputs $(j,(M,\sigma))$. Adversary E outputs 1 if $(j,(M,\sigma))$ is successful and otherwise it outputs 0. We have

$$\begin{aligned}\Pr\left[\,\mathsf{SKE}^{E}_{\mathsf{I},\mathsf{J}} \Rightarrow \mathsf{true} \mid b=1\,\right] &= \Pr\left[\,\mathrm{G}^{A}_{2} \Rightarrow \mathsf{true}\,\right]\\ \Pr\left[\,\mathsf{SKE}^{E}_{\mathsf{I},\mathsf{J}} \Rightarrow \mathsf{true} \mid b=0\,\right] &= \Pr\left[\,\mathrm{G}^{A}_{3} \Rightarrow \mathsf{true}\,\right]\end{aligned}$$

Subtracting, we get equation (15).

Next, we show how to build adversary B against game $\mathsf{INDCPA}_{\mathsf{SE}}$. Adversary B begins by executing the code of the **Initialize** procedure of G_3, thereby defining for itself all the variables there. It then starts running A on input pk. It answers A's queries to its **Next**, **Refresh**, **Expose** and **Sign** oracles exactly as G_3 does, and answers queries to the **UConv** and **RConv** oracles via the following procedures:

procedure **UConv**()
If $(\, d^{I}_{(a+1).1} = \perp \wedge d^{J}_{(a+1).1} = \perp\,)$ then
  Parse $\omega^{H}_{a.RN}$ as $\omega^{J}_{a.RN} \parallel r$
  $(\mathrm{Conv}, K_1, d^{I}_{(a+1).1}, K_1, d^{J}_{(a+1).1}) \xleftarrow{\$} \mathbf{Run}(\mathsf{I},\varepsilon,\mathsf{J},\varepsilon,;\omega^{U}_{a.RN},\omega^{J}_{a.RN})$
  $C \leftarrow \mathsf{SEnc}(K_1,(hsku_{a+1},hskr_{(a+1).1});r)$
  If $(\, a+1 = g \wedge ExpU_{a.RN})$ then $C \xleftarrow{\$} \mathbf{LR}((hsku_{a+1},hskr_{(a+1).1}),0^{|hsku_{a+1},hskr_{(a+1).1}|})$
Return $(\mathrm{Conv} \parallel C, d^{I}_{(a+1).1}, d^{J}_{(a+1).1})$

procedure **RConv**()
If $(\, d^{I}_{a.(r+1)} = \perp \wedge d^{J}_{a.(r+1)} = \perp\,)$ then
  Parse $\omega^{H}_{a.r}$ as $\omega^{J}_{a.r} \parallel r$
  $(\mathrm{Conv}, K_1, d^{I}_{a.(r+1)}, K_1, d^{J}_{a.(r+1)}) \xleftarrow{\$} \mathbf{Run}(\mathsf{I},\varepsilon,\mathsf{J},\varepsilon;\omega^{U}_{a.r},\omega^{J}_{a.r})$
  $C \leftarrow \mathsf{SEnc}(K_1,hskr_{a.(r+1)};r)$
  If $(\, a = g \wedge (\,\exists s, 1 \le s \le RN, ExpU_{a.s}\,))$ then $C \xleftarrow{\$} \mathbf{LR}(hskr_{a.(r+1)},0^{|hskr_{a.(r+1)}|})$
Return $(\mathrm{Conv} \parallel C, d^{I}_{a.(r+1)}, d^{J}_{a.(r+1)})$

In the 5th line of each procedures above, B queries its **LR** oracle. Finally, A outputs $(j,(M,\sigma))$. Adversary B outputs 1 if the forgery is a successful one, and otherwise it outputs 0. We have

$$\begin{aligned}\Pr\left[\,\mathsf{INDCPA}^{B}_{\mathsf{SE}} \Rightarrow \mathsf{true} \mid b=0\,\right] &= \Pr\left[\,\mathrm{G}^{A}_{3} \Rightarrow \mathsf{true}\,\right]\\ \Pr\left[\,\mathsf{INDCPA}^{B}_{\mathsf{SE}} \Rightarrow \mathsf{true} \mid b=1\,\right] &= \Pr\left[\,\mathrm{G}^{A}_{4} \Rightarrow \mathsf{true}\,\right]\end{aligned}$$

Subtracting, we get equation (16).

Finally, we show how adversary S works against game IRS. Given input pk, S begins by executing the code of **Initialize** procedure of G_4, defining all the variables there. It then starts running A on input pk. It answers A's queries to the **UConv**, **RConv** and **Expose** oracles via the following procedures:

procedure **Expose**(Type)
If (Type = "U") then
  $ExpU_{a.r} \leftarrow \mathsf{true}$

If $(\ a = g\ ) \vee (\ a < g \wedge ExpH_{a.r}\ )$ then Return $\perp$
$usk_{a.r} \leftarrow$ **Expose**("U") ; $hsku_a \leftarrow$ **Expose**("P") ; $hskr_{a.r} \leftarrow$ **Expose**("R") ; $uck_{a.r} \leftarrow \varepsilon$
Return $(usk_{a.r}, hsku_a, hskr_{a.r}, uck_{a.r}, \omega^U_{a.r})$
If (Type = "H") then
$ExpH_{a.r} \leftarrow \mathsf{true}$
If $(r \neq 1 \wedge ExpU_{a.r-1})$ then $ExpU_{a.r} \leftarrow \mathsf{true}$
If $(r = 1 \wedge ExpU_{a-1.RN})$ then $ExpU_{a.r} \leftarrow \mathsf{true}$
If $(\ a \leq g \wedge ExpU_{a.r}\ )$ then Return $\perp$
$hsk_{a.r} \leftarrow$ **Expose**("H") ; $hsku_a \leftarrow$ **Expose**("P") ; $hskr_{a.r} \leftarrow$ **Expose**("R") ; $hck_{a.r} \leftarrow \varepsilon$
Return $(hsk_{a.r}, hsku_a, hskr_{a.r}, hck_{a.r}, \omega^H_{a.r})$
Return $\perp$

In the 4th and 11th lines of the code above, S queries its **Expose** oracle.

procedure UConv()
If $(\ d^I_{a+1} = \perp \wedge d^J_{a+1} = \perp\ )$ then
Parse $\omega^H_{a.RN}$ as $\omega^J_{a.RN} \parallel r$
$(\mathrm{Conv}, K_1, d^I_{a+1}, K_1, d^J_{a+1}) \leftarrow$ **Run**$(\mathsf{I}, \varepsilon, \mathsf{J}, \varepsilon; \omega^U_a, \omega^J_a)$
Next()
If $(\ a + 1 = g\ \wedge ExpU_{a.RN})$ then $K_0 \xleftarrow{\$} \{0,1\}^k$; $C \xleftarrow{\$} \mathsf{SEnc}(K_0, 0^{|(hsku_{a+1}, hskr_{(a+1).1})|})$
else $(hsku_{a+1}, hskr_{(a+1).1}) \leftarrow$ **Expose**("P") ; $C \leftarrow \mathsf{SEnc}(K_1, (hsku_{a+1}, hskr_{(a+1).1}); r)$
Return $(\mathrm{Conv} \parallel C, d^I_{a+1}, d^J_{a+1})$

In the 4th line of the code above, S queries its **Next** oracle, and in the 6th line of the code above, S queries its **Expose** oracle.

procedure RConv()
If $(\ d^I_{a.(r+1)} = \perp \wedge d^J_{a.(r+1)} = \perp\ )$ then
Parse $\omega^H_{a.r}$ as $\omega^J_{a.r} \parallel r$
$(\mathrm{Conv}, K_1, d^I_{a.(r+1)}, K_1, d^J_{a.(r+1)}) \xleftarrow{\$}$ **Run**$(\mathsf{I}, \varepsilon, \mathsf{J}, \varepsilon; \omega^U_{a.r}, \omega^J_{a.r})$
Refresh()
If $(\ a = g\ \wedge (\ \exists s, 1 \leq s \leq RN, ExpU_{a.s}\ ))$ then $K_0 \xleftarrow{\$} \{0,1\}^k$; $C \xleftarrow{\$} \mathsf{SEnc}(K_0, 0^{|hskr_{a.(r+1)}|})$
else $hskr_{a.(r+1)} \leftarrow$ **Expose**("R") ; $C \leftarrow \mathsf{SEnc}(K_1, hskr_{a.(r+1)}; r)$
Return $(\mathrm{Conv} \parallel C, d^I_{a.(r+1)}, d^J_{a.(r+1)})$

In the 4th line of the code above, S queries its **Refresh** oracle, and in the 6th line of the code above, S queries its **Expose** oracle. In addition, S answers A's queries to the **Next** oracle by incrementing a and selecting $\omega^U_{a.1}, \omega^H_{a.1} \in \mathsf{COINS}$ independently at random. Similarly, S answers A's queries to the **Refresh** oracle by incrementing r and selecting $\omega^U_{a.r}, \omega^H_{a.r} \in \mathsf{COINS}$ independently at random.It answers A's queries to the **Sign** oracle via its own corresponding oracles. Finally, A outputs $(j, (M, \sigma))$ and S returns this same output. From the above, we know that if A wins game G_4, the event of $g = j$ must happen. In addition, S never queries its own **Expose** oracle which makes its forgery unsuccessful. Thus S can output the same signature $(g, (M, \sigma))$ as A and win the game IRS. So we have equation (17). ▌

References

[1] R. Anderson, Two Remarks on Public-Key Cryptology. Manuscript, 2000, and Invited Lecture at the Fourth Annual Conference on Computer and Communications Security, Zurich, Switzerland, April 1997.

[2] M. Bellare and S. Miner. A forward-secure digital signature scheme. *Advances in Cryptology – CRYPTO '99*, Lecture Notes in Computer Science Vol. 1666, M. Wiener ed., Springer-Verlag, 1999.

[3] M. Bellare, S. Duan and A. Palacio. Key Insulation and Intrusion Resilience Over a Public Channel. *Topics in Cryptology – CT-RSA '09*, Lecture Notes in Computer Science Vol. ?? , M. Fischlin ed., Springer-Verlag, 2009.

[4] M. Bellare and A. Palacio. Protecting against Key Exposure: Strongly Key-Insulated Encryption with Optimal Threshold. *Applicable Algebra in Engineering, Communication and Computing*, Vol. 16, No. 6, Springer-Verlag, 2006, pp. 379–396.

[5] M. Bellare and P. Rogaway. Entity Authentication and key distribution. *Advances in Cryptology – CRYPTO '93*, Lecture Notes in Computer Science Vol. 773, D. Stinson ed., Springer-Verlag, 1993.

[6] M. Bellare and P. Rogaway. The Security of Triple Encryption and a Framework for Code-Based Game-Playing Proofs. *Advances in Cryptology – EUROCRYPT '06*, Lecture Notes in Computer Science Vol. 4004, S. Vaudenay ed., Springer-Verlag, 2006

[7] M. Bellare and B. Yee. Forward-Security in Private-Key Cryptography. *Topics in Cryptology – CT-RSA '03*, Lecture Notes in Computer Science Vol. 2612, M. Joye ed., Springer-Verlag, 2003.

[8] M. Ben-Or, S. Goldwasser and A. Wigderson. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation. *Proceedings of the 30th Annual Symposium on the Theory of Computing*, ACM, 1998.

[9] R. Canetti, U. Feige, O. Goldreich, and M. Naor. Adaptively Secure Multi-Party Computation. *Proceedings of the 28th Annual Symposium on the Theory of Computing*, ACM, 1996.

[10] R. Canetti, S. Halevi and J. Katz. A Forward-Secure Public-Key Encryption Scheme. *Advances in Cryptology – EUROCRYPT '03*, Lecture Notes in Computer Science Vol. 2656, E. Biham ed., Springer-Verlag, 2003.

[11] R. Canetti and H. Krawczyk. Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels. *Advances in Cryptology – EUROCRYPT '01*, Lecture Notes in Computer Science Vol. 2045, B. Pfitzmann ed., Springer-Verlag, 2001.

[12] R. Canetti and H. Krawczyk. Universally Composable Notions of Key Exchange and Secure Channels. *Advances in Cryptology – EUROCRYPT '02*, Lecture Notes in Computer Science Vol. 2332, L. Knudsen ed., Springer-Verlag, 2002.

[13] Y. Dodis, M. Franklin, J. Katz, A. Miyaji and M. Yung. Intrusion-Resilient Public-Key Encryption. *Topics in Cryptology – CT-RSA '03*, Lecture Notes in Computer Science Vol. 2612, M. Joye ed., Springer-Verlag, 2003.

[14] Y. Dodis, M. Franklin, J. Katz, A. Miyaji and M. Yung. A Generic Construction for Intrusion-Resilient Public-Key Encryption. *Topics in Cryptology – CT-RSA '04*, Lecture Notes in Computer Science Vol. 2964, T. Okamoto ed., Springer-Verlag, 2004.

[15] Y. Dodis, J. Katz, S. Xu and M. Yung. Key-Insulated Public Key Cryptosystems. *Advances in Cryptology – EUROCRYPT '02*, Lecture Notes in Computer Science Vol. 2332, L. Knudsen ed., Springer-Verlag, 2002.

[16] Y. Dodis, J. Katz, S. Xu and M. Yung. Strong Key-Insulated Signature Schemes. *Public-Key Cryptography '03*, Lecture Notes in Computer Science Vol. 2567, Y. Desmdedt ed., Springer-Verlag, 2003.

[17] C. Dwork, M. Naor, O. Reingold and L. Stockmeyer. Magic Functions. *Proceedings of the 40th Symposium on Foundations of Computer Science*, IEEE, 1999.

[18] G. Hanaoka, Y. Hanaoka and H. Imai. Parallel Key-Insulated Public Key Encryption. *Public-Key Cryptography '06*, Lecture Notes in Computer Science Vol. 3958, M. Yung, Y. Dodis, A. Kiayias and T. Malkin ed., Springer-Verlag, 2006.

[19] Y. Hanaoka, G. Hanaoka, J. Shikata, H. Imai. Identity-based Heirarchical Strongly Key-Insulated Encryption and its Application. *Advances in Cryptology – ASIACRYPT '05*, Lecture Notes in Computer Science Vol. 3788, B. Roy ed., Springer-Verlag, 2005.

[20] G. Itkis and L. Reyzin. SiBIR: Signer-Base -Resilient Signatures. *Advances in Cryptology – CRYPTO '02*, Lecture Notes in Computer Science Vol. 2442, M. Yung ed., Springer-Verlag, 2002.

procedure Initialize
$(pk, usk_{0.0}, hsk_{0.0}) \xleftarrow{\$} \mathsf{KG}$
$(uck_{0.0}, hck_{0.0}) \xleftarrow{\$} \mathsf{CKG}$
$(hsk_{1.0}, hsku_1) \xleftarrow{\$} \mathsf{HKU}(1, pk, hsk_{0.0})$
$(hsk_{1.1}, hskr_{1.1}) \xleftarrow{\$} \mathsf{HKR}(1, pk, hsk_{1.0})$
$\omega^U_{0.0} \xleftarrow{\$} \mathsf{COINS}$; $\omega^H_{0.0} \xleftarrow{\$} \mathsf{COINS}$
$St^U_{1.1} \leftarrow (usk_{0.0}, uck_{0.0}, \omega^U_{0.0})$
$St^H_{1.1} \leftarrow (hsku_1, hskr_{1.1}, hck_{0.0}, \omega^H_{0.0})$
$a \leftarrow 0$; $r \leftarrow 0$; $S \leftarrow \emptyset$
$g \xleftarrow{\$} \{1, \ldots, q_U\}$
Return pk

procedure Refresh()
$r \leftarrow r + 1$
If ($d^U_{a.r} = \mathsf{acc}$) then
  $(hskr_{a.r}, uck_{a.r}) \leftarrow St^U_{a.r}$; $\omega^U_{a.r} \xleftarrow{\$} \mathsf{COINS}$
  $usk_{a.r} \leftarrow \mathsf{UKR}(a, pk, hskr_{a.r}, usk_{a.(r-1)})$
  If ($r = RN$) then
    $St^U_{(a+1).1} \leftarrow (usk_{a.r}, uck_{a.r}, \omega^U_{a.r})$
  Else
    $St^U_{a.(r+1)} \leftarrow (usk_{a.r}, uck_{a.r}, \omega^U_{a.r})$
If ($d^H_{a.r} = \mathsf{acc}$) then
  $hck_{a.r} \leftarrow St^H_{a.r}$; $\omega^H_{a.r} \xleftarrow{\$} \mathsf{COINS}$
  If ($r = RN$) then
    $(hsk_{(a+1).0}, hsku_{a+1}) \xleftarrow{\$} \mathsf{HKU}(a+1, pk, hsk_{a.r})$
    $(hsk_{(a+1).1}, hskr_{(a+1).1}) \xleftarrow{\$} \mathsf{HKR}(a+1, pk, hsk_{(a+1).0})$
    $St^H_{(a+1).1} \leftarrow (hsku_{a+1}, hskr_{(a+1).1}, hck_{a.r}, \omega^H_{a.r})$
  Else
    $(hsk_{a.(r+1)}, hskr_{a.(r+1)}) \xleftarrow{\$} \mathsf{HKR}(a, pk, hsk_{a.r})$
    $St^H_{a.(r+1)} \leftarrow (hskr_{a.(r+1)}, hck_{a.r}, \omega^H_{a.r})$

procedure Sign(M)
$(a, \sigma) \xleftarrow{\$} \mathsf{Sig}(a, usk_{a.r}, M)$
$S \leftarrow S \cup \{(a, M)\}$
Return (a, σ)

procedure UConv() // G_0,G_1,G_2
If ($d^I_{(a+1).1} = \perp \wedge d^J_{(a+1).1} = \perp$) then
  Parse $\omega^H_{a.RN}$ as $\omega^J_{a.RN} \parallel r$
  $(\mathrm{Conv}, K_1, d^I_{(a+1).1}, K_1, d^J_{(a+1).1}) \xleftarrow{\$} \mathbf{Run}(\mathsf{I}, \varepsilon, \mathsf{J}, \varepsilon; \omega^U_{a.RN}, \omega^J_{a.RN})$
  $C \leftarrow \mathsf{SEnc}(K_1, (hsku_{a+1}, hskr_{(a+1).1}); r)$
  $St^U_{(a+1).1} \leftarrow (hsku_{a+1}, hskr_{(a+1).1}, \varepsilon)$; $St^H_{(a+1).1} \leftarrow \varepsilon$
Return $(\mathrm{Conv} \parallel C, d^I_{(a+1).1}, d^J_{(a+1).1})$

procedure RConv() // G_0,G_1,G_2
If ($d^I_{a.(r+1)} = \perp \wedge d^J_{a.(r+1)} = \perp$) then
  Parse $\omega^H_{a.r}$ as $\omega^J_{a.r} \parallel r$
  $(\mathrm{Conv}, K_1, d^I_{a.(r+1)}, K_1, d^J_{a.(r+1)}) \xleftarrow{\$} \mathbf{Run}(\mathsf{I}, \varepsilon, \mathsf{J}, \varepsilon; \omega^U_{a.r}, \omega^J_{a.r})$
  $C \leftarrow \mathsf{SEnc}(K_1, hskr_{a.(r+1)}; r)$
  $St^U_{a.(r+1)} \leftarrow (hskr_{a.(r+1)}, \varepsilon)$; $St^H_{a.(r+1)} \leftarrow \varepsilon$
Return $(\mathrm{Conv} \parallel C, d^I_{a.(r+1)}, d^J_{a.(r+1)})$

procedure Expose(Type) // G_0, G_1, $\boxed{G_2}$, $\boxed{G_3}$, $\boxed{G_4}$
$x \leftarrow \perp$
If (Type = "U") then
  $ExpU_{a.r} \leftarrow \mathsf{true}$
  $x \leftarrow (usk_{a.r}, hsku_a, hskr_{a.r}, uck_{a.r}, \omega^U_{a.r})$
  If ($a = g$)$\vee$($a < g \wedge ExpH_{a.r}$) then $\boxed{x \leftarrow \perp}$
If (Type = "H") then
  $ExpH_{a.r} \leftarrow \mathsf{true}$
  $x \leftarrow (hsk_{a.r}, hsku_a, hskr_{a.r}, hck_{a.r}, \omega^H_{a.r})$
  If ($r \neq 1 \wedge ExpU_{a.r-1}$) then $ExpU_{a.r} \leftarrow \mathsf{true}$
  If ($r = 1 \wedge ExpU_{a-1.RN}$) then $ExpU_{a.r} \leftarrow \mathsf{true}$
  If ($a \leq g \wedge ExpU_{a.r}$) then $\boxed{x \leftarrow \perp}$
Return x

procedure Next()
$a \leftarrow a + 1$
If ($d^U_{a.1} = \mathsf{acc}$) then
  $(hsku_a, hskr_{a.1}, uck_{a.1}) \leftarrow St^U_{a.1}$; $\omega^U_{a.1} \xleftarrow{\$} \mathsf{COINS}$
  $usk_{a.0} \leftarrow \mathsf{UKU}(a, pk, hsku_a, usk_{(a-1).r})$
  $usk_{a.1} \leftarrow \mathsf{UKR}(a, pk, hskr_{a.1}, usk_{a.0})$
  $St^U_{a.2} \leftarrow (usk_{a.1}, uck_{a.1}, \omega^U_{a.1})$
If ($d^H_{a.1} = \mathsf{acc}$) then
  $hck_{a.1} \leftarrow St^H_{a.1}$; $\omega^H_{a.1} \xleftarrow{\$} \mathsf{COINS}$
  $(hsk_{a.2}, hskr_{a.2}) \xleftarrow{\$} \mathsf{HKR}(a, pk, hsk_{a.1})$
  $St^H_{a.2} \leftarrow (hskr_{a.2}, hck_{a.1}, \omega^H_{a.1})$
$r \leftarrow 1$

procedure Finalize$(M, (j, \sigma))$ // G_0, $\boxed{G_1}$, $\boxed{G_2}$, $\boxed{G_3}$, $\boxed{G_4}$
If $((j, M) \in S \vee \mathsf{Ver}(pk, M, (j, \sigma)) = 0)$ then Return 0
If $\exists s (1 \leq s \leq RN \wedge ExpU_{j.s})$ then Return 0
If $\exists i \exists s (1 \leq i < j \wedge 1 \leq s \leq RN_i \wedge ExpU_{i.s} \wedge$
  $ExpH_{i.s})$ then Return 0
Return $\boxed{g = j \wedge}$ 1

Figure 8: Games used to used to prove security of public-channel intrusion resilience.

procedure **UConv**() // G_3
If ($d^I_{(a+1).1} = \perp \wedge d^J_{(a+1).1} = \perp$) then
  Parse $\omega^H_{a.RN}$ as $\omega^J_{a.RN} \parallel r$
  $(\text{Conv}, K_1, d^I_{(a+1).1}, K_1, d^J_{(a+1).1}) \xleftarrow{\$} \mathbf{Run}(\mathsf{I}, \varepsilon, \mathsf{J}, \varepsilon, ; \omega^U_{a.RN}, \omega^J_{a.RN})$
  $C \leftarrow \mathsf{SEnc}(K_1, (hsku_{a+1}, hskr_{(a+1).1}); r)$
  If ($a + 1 = g \wedge ExpU_{a.RN}$) then
    $K_0 \xleftarrow{\$} \{0,1\}^k$; $C \xleftarrow{\$} \mathsf{SEnc}(K_0, (hsku_{a+1}, hskr_{(a+1).1}))$
  $St^U_{(a+1).1} \leftarrow (hsku_{a+1}, hskr_{(a+1).1}, \varepsilon)$; $St^H_{(a+1).1} \leftarrow \varepsilon$
Return (Conv $\parallel C, d^I_{(a+1).1}, d^J_{(a+1).1}$)

procedure **RConv**() // G_3
If ($d^I_{a.(r+1)} = \perp \wedge d^J_{a.(r+1)} = \perp$) then
  Parse $\omega^H_{a.r}$ as $\omega^J_{a.r} \parallel r$
  $(\text{Conv}, K_1, d^I_{a.(r+1)}, K_1, d^J_{a.(r+1)}) \xleftarrow{\$} \mathbf{Run}(\mathsf{I}, \varepsilon, \mathsf{J}, \varepsilon; \omega^U_{a.r}, \omega^J_{a.r})$
  $C \leftarrow \mathsf{SEnc}(K_1, hskr_{a.(r+1)}; r)$
  If ($a = g \wedge (\exists s, 1 \le s \le RN, ExpU_{a.s})$) then
    $K_0 \xleftarrow{\$} \{0,1\}^k$; $C \xleftarrow{\$} \mathsf{SEnc}(K_0, hskr_{a.(r+1)})$
  $St^U_{a.(r+1)} \leftarrow (hskr_{a.(r+1)}, \varepsilon)$; $St^H_{a.(r+1)} \leftarrow \varepsilon$
Return (Conv $\parallel C, d^I_{a.(r+1)}, d^J_{a.(r+1)}$)

procedure **UConv**() // G_4
If ($d^I_{(a+1).1} = \perp \wedge d^J_{(a+1).1} = \perp$) then
  Parse $\omega^H_{a.RN}$ as $\omega^J_{a.RN} \parallel r$
  $(\text{Conv}, K_1, d^I_{(a+1).1}, K_1, d^J_{(a+1).1}) \xleftarrow{\$} \mathbf{Run}(\mathsf{I}, \varepsilon, \mathsf{J}, \varepsilon, ; \omega^U_{a.RN}, \omega^J_{a.RN})$
  $C \leftarrow \mathsf{SEnc}(K_1, (hsku_{a+1}, hskr_{(a+1).1}); r)$
  If ($a + 1 = g \wedge ExpU_{a.RN}$) then
    $K_0 \xleftarrow{\$} \{0,1\}^k$; $C \xleftarrow{\$} \mathsf{SEnc}(K_0, 0^{|hsku_{a+1}, hskr_{(a+1).1}|})$
  $St^U_{(a+1).1} \leftarrow (hsku_{a+1}, hskr_{(a+1).1}, \varepsilon)$; $St^H_{(a+1).1} \leftarrow \varepsilon$
Return (Conv $\parallel C, d^I_{(a+1).1}, d^J_{(a+1).1}$)

procedure **RConv**() // G_4
If ($d^I_{a.(r+1)} = \perp \wedge d^J_{a.(r+1)} = \perp$) then
  Parse $\omega^H_{a.r}$ as $\omega^J_{a.r} \parallel r$
  $(\text{Conv}, K_1, d^I_{a.(r+1)}, K_1, d^J_{a.(r+1)}) \xleftarrow{\$} \mathbf{Run}(\mathsf{I}, \varepsilon, \mathsf{J}, \varepsilon; \omega^U_{a.r}, \omega^J_{a.r})$
  $C \leftarrow \mathsf{SEnc}(K_1, hskr_{a.(r+1)}; r)$
  If ($a = g \wedge (\exists s, 1 \le s \le RN, ExpU_{a.s})$) then
    $K_0 \xleftarrow{\$} \{0,1\}^k$; $C \xleftarrow{\$} \mathsf{SEnc}(K_0, 0^{|hskr_{a.(r+1)}|})$
  $St^U_{a.(r+1)} \leftarrow (hskr_{a.(r+1)}, \varepsilon)$; $St^H_{a.(r+1)} \leftarrow \varepsilon$
Return (Conv $\parallel C, d^I_{a.(r+1)}, d^J_{a.(r+1)}$)

Figure 9: Procedures **UConv**(), **RConv**() of games G_3 and G_4.