Forward-Security under Continual Leakage

Mihir Bellare[1] Adam O'Neill[2] Igors Stepanovs[3]

May 2017

Abstract

Current signature and encryption schemes secure against continual leakage fail completely if the key in any time period is fully exposed. We suggest forward security as a second line of defense, so that in the event of full exposure of the current secret key, at least uses of keys prior to this remain secure, a big benefit in practice. (For example if the signer is a certificate authority, full exposure of the current secret key would not invalidate certificates signed under prior keys.) We provide definitions for signatures and encryption that are forward-secure under continual leakage. Achieving these definitions turns out to be challenging, and we make initial progress with some constructions and transforms.

[1] Department of Computer Science & Engineering, University of California San Diego, 9500 Gilman Drive, La Jolla, California 92093, USA. Email: `mihir@eng.ucsd.edu`. URL: `http://cseweb.ucsd.edu/~mihir/`. Supported in part by NSF grants CNS-1526801 and CNS-1228890, ERC Project ERCC FP7/615074 and a gift from Microsoft.

[2] Department of Computer Science, Georgetown University, 3700 Reservoir Road NW, Washington, DC 20057, USA. Email: `adam@cs.georgetown.edu`. URL: `http://people.cs.georgetown.edu/~adam/`.

[3] Department of Computer Science & Engineering, University of California San Diego, 9500 Gilman Drive, La Jolla, California 92093, USA. Email: `istepano@eng.ucsd.edu`. URL: `https://cseweb.ucsd.edu/~istepano/`. Supported in part by grants of first author.

Contents

1 Introduction **2**

2 Preliminaries **6**

3 Forward security under continual leakage **7**

4 FUFCL signatures from UFCL signatures **10**

5 A unified paradigm for constructing FS+CL schemes **12**
5.1 FUFCL signatures from FOWCL key-evolution schemes 15
5.2 FINDCL encryption from FOWCL key-evolution schemes 20

A Construction of KES-TREE **26**

B Proof of Theorem 4.1 **29**

1 Introduction

Classically, cryptography assumes secure endpoints and an insecure communication channel. Malware and sidechannel attacks bring the threat to the endpoints: Information about keys stored on our system can be leaked or exfiltrated to the adversary. Let us begin by reviewing two ways to address this for public-key cryptography, namely forward security and leakage resilience.

<u>Forward security.</u> The threat of exposure of a secret (signing or decryption) key due to compromise of the system storing the key is not new. Forward security (FS) was developed in the late 1990s as a way to mitigate the damage. The idea of forward secure signatures was suggested by Anderson [5] and formalized by Bellare and Miner (BM) [8]. Later Canetti, Halevi and Katz (CHK) [15] formalized forward secure encryption. Subsequent work gave many schemes and extensions.

Forward-security [5, 8] introduced the *key evolution* paradigm: evolve the secret key over time while keeping the public key fixed. At time period i, the secret key is sk_i, and the signing algorithm (we will discuss signatures rather than encryption as an example), applied to it and a message m, produces a signature that is a pair (i, σ), meaning the time period is explicitly included in the signature. At the start of time period $i + 1$, a (public) update function is applied to sk_i to get sk_{i+1}, and sk_i is deleted from the system. An attack compromising the system at some time i obtains sk_i, and automatically sk_j for any $j \geq i$ since the update function is public. Security of signatures as defined in BM [8] required that possession of sk_i does not allow forgery of signatures with time period j —meaning ones of the form (j, σ)— for $j < i$. It follows that possession of sk_i does not allow recovery of sk_j for $j < i$, meaning the update function must be one-way.

<u>The CA example.</u> What does FS buy us? An illustrative example is when the signer is a certificate authority (CA). Assume the CA creates certificates using a normal (not forward secure) signature scheme, with its (single, static) secret key sk. Say that in time period 1, it creates a certificate (m, σ) for Alice, where m is Alice's certificate data (her pubic key and so on) and σ is the CA's signature on m. Suppose, in time period 168, the CA system is infiltrated by malware (a realistic possibility) and sk is exposed. Discovering this, the CA immediately revokes its public key. Now suppose in time period 169, Bob receives from Alice the (valid!) certificate (m, σ) in a TLS connection. Finding the CA public key on his revocation list, he will reject certificate (m, σ) as invalid and deny the TLS connection. Security-wise, this is the right and necessary thing to do: there is no way for Bob to know that (m, σ) is not a forgery. But the cost is enormous: all certificates the CA had issued prior to the revocation (which could be many years worth of certificates) must be discarded, and many TLS connections will be rejected, causing serious disruption to web services. Time-stamping the signature will not fix this, since, once the adversary has the secret key, it can forge the time-stamp too.

But now suppose the CA used a forward secure signature scheme instead of a normal one, so that the January 1st signature has the form $(1, \sigma)$, Alice's certificate thus being $(m, (1, \sigma))$. The infiltration in time period 168 exposes secret key sk_{168}. As before, the CA revokes its public key, and now we consider Bob receiving the certificate $(m, (1, \sigma))$ in time period 169. He sees the CA public key on his revocation list, but he also sees the revocation is marked with time period $168 > 1$. Now he can *safely accept* the certificate $(m, (1, \sigma))$ and proceed with the TLS connection, because forward-security guarantees that $(m, (1, \sigma))$ cannot be a forgery. That is, certificates created prior to the exposure are still secure and valid. This is a significant advantage in the event of compromise.

Note that FS does not prevent (or even make more difficult) exposure of a secret key. That is not its aim. Its aim is to mitigate the damage caused by an exposure, if and when the latter occurs.

<u>Leakage-resilience.</u> Motivated by sidechannel attacks, leakage resilience aims to preserve security even if some information $f(sk)$ about the secret key sk is leaked. In the bounded memory

leakage model of Akavia et al. [2] and extensions [38, 21], f is any function returning a number of bits enough short of the length $|sk|$ of sk. However if the adversary has some sidechannel capability, it may, over time, gather enough bits to expose the entire key, and then security is lost. To protect against this, Dodis et al. [19] and Brakerski et al. [14] propose the continual leakage (CL) model. As in FS, the secret key is updated in each time period while the public key stays fixed. In each time period i, the adversary may obtain a bounded amount of leakage $f_i(sk_i)$ on the current secret key, yet security must be maintained. The gain is that the sidechannel attack has limited time to attack a particular key before it is updated, and once that happens it must effectively start from scratch.

Security is parameterized by a leakage rate, the scheme being δ-CL if it remains secure when the number of bits leaked in any period is restricted to at most a δ fraction of the length of the secret key. Achieving δ-CL-security is not easy. One subtlety is that the update function, unlike for FS, must be randomized. Secure schemes have been provided in [19, 14] for the cases of encryption and signatures, while other works looked at extensions to basic notions and treated other primitives [31, 11, 33, 37, 34, 27, 4, 18, 39, 17].

<u>The problem.</u> Security in the CL model relies on the assumption that the amount of leakage in a particular time period is bounded, in particular short of the length of the key itself. If the entire key is leaked in some time period, security is lost entirely. One could make updates more frequent to restrict the time the attacker has to expose a key before it is updated, but, while this may be reasonable for certain kinds of side-channel attacks, it may not be effective when the attack is malware on your system that can directly exfiltrate the key. Also it is not clear how to pick an update frequency or evaluate the security benefits of a choice. We introduce forward-security under continual leakage as a way to maintain the CL guarantee but add a second line of defense against full key exposure via FS.

<u>Forward-security under continual leakage.</u> We continue, as with both FS and CL, to work in the model where the public key is fixed but the secret key evolves with time, a public update function being applied to the period i key sk_i to produce the next key sk_{i+1}. The first definition one might consider is to ask that the scheme be both (1) CL-secure, and (2) FS-secure. We can do better. We ask that FS holds even under CL, in the following sense. In our game, in any time period, the adversary get a bounded amount of leakage $f_i(sk_i)$ on the key sk_i in that time period i, just as in CL. Additionally, it can, in some time period i of its choice, expose and obtain the *entire* secret key sk_i. The requirement is that of FS, namely security of usages of keys sk_j for $j < i$. Note a FS+CL scheme defined in this way is both CL (restrict attention to adversaries that do not make the full expose query) and FS (restrict attention to adversaries that do make this query but do not leak any information on prior keys). But it requires more than the two individually because security of keys sk_j for $j < i$ is guaranteed even when sk_i is known to the adversary and the adversary has leakage on all the keys sk_j. As with CL, security is parameterized by a leakage rate δ.

Within this template, the precise definition of security depends on the primitive. In Section 3, we define key-evolving signature schemes and a notion of δ-FUFCL security, for Forward Unforgeability under Continual Leakage. The definition is parameterized by the leakage rate δ. We also define key-evolving encryption schemes and a notion of δ-FINDCL security, for Forward INDistinguishability under Continual Leakage. As a tool in obtaining security against continual leakage, Dodis et al. [19] introduced the notion of relations that are one-way under CL, and in analogy, as a tool to obtain forward security under continual leakage, we define in Section 5 the notion of a δ-FOWCL relation, for Forward One-Wayness under Continual Leakage.

Goal	$\delta' =$	Assumptions	Section
δ'-FUFCL signatures	$\delta/(d+1)$	δ-CL signatures	4
δ'-FUFCL signatures	δ	δ-FOWCL KE for T periods + WS	5.1
δ'-FINDCL encryption	δ	δ-FOWCL KE for T periods + WE	5.2

Figure 1: Proposed constructions of δ'-FUFCL key-evolving signature schemes, and δ'-FINDCL key-evolving encryption schemes for different values of continual-leakage fraction δ' and for T time periods. We assume that $\mathsf{T} = 2^d$ for some $d \in \mathbb{N}$.

THE BENEFITS. To see the benefits provided by forward security under continual leakage over CL security alone, let us return to the CA example discussed above. Suppose that the CA is concerned about leakage and uses a CL-secure signature scheme in place of the normal secure signature scheme. Alice's certificate, produced in time period 1, has the form $(m, (1, \sigma))$. Now suppose, due to malware on the system in time period 168, the secret key sk_{168} is exposed, and the public key revoked. Bob receives the certificate $(m, (1, \sigma))$ for a TLS connection in time period 169. Seeing the CA's public key on the certificate revocation list, he cannot accept Alice's certificate, because, in a CL scheme, possession of sk_{168} could allow forgery of signatures for time period 1. Thus millions of certificates, issued over years by the CA, suddenly become obsolete. The cost in disruption to web services (`gmail`, `amazon`, ...) using TLS is huge. However if the scheme is FS+CL secure, Bob can in confidence accept Alice's certificate, because the revocation period is $168 > 1$. Thus we have provided leakage resilience with a second line of defense that significantly mitigates damage caused by full key exposures.

CHALLENGES. We would like to give FS+CL schemes for both signatures and encryption, meaning a δ-FUFCL signature scheme and a δ-FINDCL encryption scheme. This is surprisingly challenging. The first thought one may have is that perhaps some existing CL-secure scheme is already FS+CL. This is not true, because all existing CL schemes update their secret keys by merely re-randomizing them. So a full exposure of the key sk_i in some time period i results in full recovery of the secret keys for *all* time periods, meaning that the schemes are not even FS, let alone FS+CL. The complementary question is whether any existing FS scheme happens to be FS+CL, but this seems evidently false because existing FS schemes provide no security under leakage. One reason is that the update functions are deterministic, and no scheme with a deterministic update function can be CL secure. The latter is because otherwise an adversary can repeatedly leak bits of a secret key sk_t for some future time period t, by querying $f_i(sk_i)$ for functions f_i that use sk_i to compute sk_t.

The natural next construction approach to consider is a modular one. We have CL-secure schemes, and we have FS-secure ones. Is there some way to combine a CL scheme with an FS one to get a FS+CL one? We do not know a fully general way to do this, but our first scheme is obtained by a generic transform of this ilk, as we now discuss.

GENERIC TRANSFORM FROM CL. Our constructions are summarized in Fig. 1. Our first result is a generic transform of any CL scheme into a FS+CL one in the case of signatures. FS+CL security is proven with *no extra assumptions* beyond the CL security of the base scheme. We can now use existing CL signature schemes [14, 19]. Thus we obtain the first constructions of FUFCL signatures schemes.

Our generic transform is tree-based. To get a FS+CL signature scheme, we use the binary tree FS signature construction from BM [8] with a (any) CL scheme as the base scheme. A drawback of this transform, however, is that it degrades the relative leakage parameter: If we start with a

δ-CL scheme and the number of time periods is $\mathsf{T} = 2^d$ then we get a δ'-FS+CL scheme with $\delta' = \delta/(d+1)$. In particular we do not get a FUFCL signature scheme with constant relative leakage.

This approach does not work in the case of encryption. For example, it is tempting to start from the CL HIBE of Lewko, Rouselakis and Waters [34] and use it to build binary-tree encryption (BTE) following the construction of FS-encryption from CHK [15], but this fails. The problem is that FS+CL security of the resulting scheme requires that multiple nodes of the BTE construction can be leaked on *jointly*, whereas the CL security of HIBE only buys us leakage on each such node *individually*. We will construct a FINDCL encryption scheme in a different way that leverages both the FUFCL signature scheme we have just built and witness encryption, as we discuss next.

Transforms using Witness primitives. The second set of constructions extends the paradigm of Dodis et al. [19]. They used a key evolution scheme that is one-way under continual leakage to build CL-secure signatures and encryption. We assume a key evolution scheme that is *forward* one-way under continual-leakage (FOWCL KE). Then we present a unified paradigm to get FS+CL signatures and encryption using, as an additional tool, WX, where the "W" stands for "Witness" and X=S for signatures, and X=E for encryption. In other words, we use witness signatures (WS) [16, 7] to get FS+CL signatures, and witness encryption (WE) [26, 6, 28] to get FS+CL encryption. In this case there is no loss of relative leakage, meaning if we start with a δ-FOWCL KE scheme we get δ'-FS+CL encryption and signature schemes with $\delta' = \delta$.

To obtain FS+CL schemes from these results, we need to instantiate the components. This means we need: (1) A FOWCL key evolution scheme (2) A WS scheme to get FS+CL signatures and a WE scheme to get FS+CL encryption. Let us now look into this.

First, for (2), witness signatures as we define them in Section 5.1 are, as explained there, easily obtained from NIZKs since they are just another name for signatures of knowledge [16, 7], different from the (impossible) witness signatures of GJK [29]. In other words, these are readily available under standard assumptions. Witness encryption, as we discuss further in Section 5.2, is more difficult since we do require (a weak form of) extractability. Now turning to (1), we can get a FOWCL KE scheme by using the tree-based FUFCL signature scheme obtained via our first (generic transform) result.

The main outcome from this is the first construction of a FINDCL encryption scheme. The assumptions are any CL signature scheme (which yields a FUFCL signature scheme via our tree-based construction, and thence a FOWCL KE scheme as noted above) plus extractable witness encryption. Due to the use of our tree-based construction, the relative leakage will again degrade compared to that of the starting CL signature scheme, so we again fail to get FINDCL encryption with constant relative leakage. The extractable witness encryption we use evades the negative results of Garg et al. [25] because we only require security for a single **NP**-relation, but it is nonetheless a very strong assumption.

This paradigm can be used to get FUFCL signatures too, but with the instantiation we have of the FOWCL KE scheme itself coming from our tree-based FUFCL signature scheme constructed above, we would not not obtain anything over and above our generic transform result discussed above. But the transforms are still interesting both for signatures and encryption because if it were possible to find a FOWCL KE scheme withstanding constant relative leakage, we would immediately get a FUFCL signature scheme and a FINDCL encryption scheme with the same constant relative leakage, assuming the witness primitives. Thus they help to reduce the problem of FS+CL signatures and encryption to the single and hopefully simpler problem of FOWCL KE.

Status and perspective. None of our constructions is entirely satisfactory. Referring to Fig. 1, the drawback of the result in the 1st row is the factor $d+1$ loss in relative leakage. While we think

the transform of the 2nd row is interesting, we do not right now have a constant relative leakage FOWCL KE to use as a starting point. The drawback of the result in the 3rd row is that the WE does need some type of extractability, which is sometimes subject to negative results [25].

Our view is that we initiate the study of an interesting goal (FSCL), providing motivation and definitions. We explore natural construction approaches, such as the generic tree-based transform. We also reduce achieving FUFCL signatures and FINDCL encryption to the potentially more tractable task of achieving FOWCL relations, albeit under strong assumptions in the 2nd case. Our work indicates that full solutions in this domain are challenging, more so than they might seem. We believe the exposure of our work has value in drawing attention to these open problems, and that others will find these problems technically interesting and make more progress on them than we have been able to make.

RELATED WORK. Bounded leakage-resilience and its extensions were studied for various primitives including encryption and signature schemes in [2, 21, 38, 3, 35, 13, 20, 24]. Continual leakage-resilience (CL) was studied in [14, 19, 34]. In particular, [14, 19] provide CL signature schemes with leakage rate $1 - o(1)$ (i.e. arbitrarily close to 1) in bilinear groups. These schemes can be plugged into our generic transform described above. Extensions of the basic CL notions have been considered in [31, 11, 33, 37, 34, 27, 4, 18, 39, 17], which yield further CL schemes.

After the initial schemes of BM [8], various follow-up works constructed more efficient forward-secure signature schemes or gave other extensions, including [1, 32, 30, 10]. Malkin et al. [36] constructed forward-secure signatures for an unbounded number of time periods. Unfortunately, their framework does not allow to get FS+CL schemes by composing CL-schemes. Their composition methods require the secret key to contain various components (other secret keys, and random seeds) that remain unchanged for a number of time periods. If leakage on these parts of their secret key is allowed, then security is lost.

Leakage on updates, where leakage is allowed on the coins used in updating keys, is considered in [33, 22, 17]. Our model and results are a first step that do not allow leakage on updates. This is an interesting consideration for future work.

2 Preliminaries

NOTATION. We denote by $\lambda \in \mathbb{N}$ the security parameter and by 1^λ its unary representation. For $i \in \mathbb{N}$ we let $[i]$ denote the set $\{1, \ldots, i\}$. We let ε denote the empty string. We denote the length of a string $x \in \{0,1\}^*$ by $|x|$. By $x \,\|\, y$ we denote the concatenation of strings x, y. Algorithms may be randomized unless otherwise indicated. Running time is worst case. "PT" stands for "polynomial-time," whether for randomized algorithms or deterministic ones. If A is an algorithm, we let $y \leftarrow A(x_1, \ldots; r)$ denote running A with random coins r on inputs $x_1, \ldots$ and assigning the output to y. We let $y \leftarrow\$ A(x_1, \ldots)$ be the result of picking r at random and letting $y \leftarrow A(x_1, \ldots; r)$. We let $[A(x_1, \ldots)]$ denote the set of all possible outputs of A when invoked with inputs $x_1, \ldots$. We say that $f : \mathbb{N} \to \mathbb{R}$ is negligible if for every positive polynomial p, there exists $\lambda_p \in \mathbb{N}$ such that $f(\lambda) < 1/p(\lambda)$ for all $\lambda \geq \lambda_p$. We use the code based game playing framework of Bellare and Rogaway [9]. (See Fig. 2 for an example.) By $\mathrm{G}^{\mathcal{A}}(\lambda)$ we denote the event that the execution of game G with adversary $\mathcal{A}$ and security parameter λ results in the game returning true. Booleans are assumed initialized to false and integers to 0.

NP-RELATIONS. Relation R specifies a PT algorithm $\mathsf{R.Vf}$. Witness verification algorithm $\mathsf{R.Vf}$ takes an instance $x \in \{0,1\}^*$ and a candidate witness $w \in \{0,1\}^*$ to return a decision in $\{\mathsf{true}, \mathsf{false}\}$. For any $x \in \{0,1\}^*$ we let $\mathsf{R}(x) = \{\, w \in \{0,1\}^* \;:\; \mathsf{R.Vf}(x, w) \,\}$ be the witness set of x. We let $\mathcal{L}(\mathsf{R}) = \{\, x \in \{0,1\}^* \;:\; \mathsf{R}(x) \neq \emptyset \,\}$ be the language defined by R. We say that R is an **NP**-relation,

and $\mathcal{L}(\mathsf{R})$ an **NP** language, if there exists a witness-length polynomial $\mathsf{R.wl}: \mathbb{N} \to \mathbb{N}$ such that $\mathsf{R}(x) \subseteq \bigcup_{\ell \leq \mathsf{R.wl}(|x|)} \{0,1\}^\ell$ for all $x \in \{0,1\}^*$.

3 Forward security under continual leakage

In this section we consider key-evolving signature and encryption schemes, for which we provide definitions of forward security under continual leakage. A key-evolving scheme has a single public key pk, while its secret key evolves with time, $sk_1 \to sk_2 \to \cdots \to sk_T$, where "$\to$" is implemented by an update algorithm and T is the number of time periods supported by the scheme.

In the continual leakage setting, in every time period t the attacker can obtain leakage on the current secret key sk_t. The security requirement from prior work [19, 14] is that security under all keys be maintained. For this to be achievable, the leakage on each key must be assumed to be bounded. But this boundedness assumption is unrealistic, and in practice a key may leak entirely. In this case, prior systems require and provide no security. We propose to add forward security as a second line of defense, asking that even if a key sk_{t^*} leaks fully for any time period t^*, security under prior keys will not be compromised. This brings important gains, for example, when signing certificates, that those signed prior to the full exposure do not have to be revoked. Forward-security under continual leakage, as we define it, simultaneously implies both security under continual leakage and classical forward security.

We now define forward unforgeability under continual leakage (FUFCL) for key-evolving signature schemes, and forward indistinguishability under continual leakage (FINDCL) for key-evolving encryption schemes. Security games for both notions are in Fig. 2.

Public and secret components of a secret key. We will parameterize security notions of key-evolving schemes by a function $\delta: \mathbb{N} \to [0,1]$ that, informally, denotes the fraction of the secret key that may leak in every time period. However, the secret key may contain some information that is necessary only for the key-evolving functionality of the scheme, but is not required to be hidden for the security of the scheme. Therefore, it is not useful to consider a leakage metric that compares different schemes based on the fraction of the *entire* secret key that may be leaked per time period: this fraction might be very small just because the secret keys of a particular scheme contain a lot of information that is not required to be kept secret. We address this by requiring that all secret keys used by key-evolving schemes can be parsed as a pair (pc, sc), where pc denotes the public component of the secret key and sc denotes the secret component of the secret key. We then define δ-leakage security of key-evolving schemes to denote the fraction of the *secret component* of the secret key that may leak in every time period. Our security games provide the public components of all secret keys to the adversary for free.

Key-evolving signature schemes. A key-evolving signature scheme KES specifies PT algorithms $\mathsf{KES.Kg}$, $\mathsf{KES.Up}$, $\mathsf{KES.Sig}$ and $\mathsf{KES.Vf}$, where $\mathsf{KES.Vf}$ is deterministic. Associated to KES are the following polynomials: public-key length $\mathsf{KES.pkl}: \mathbb{N} \to \mathbb{N}$, secret-key length $\mathsf{KES.skl}: \mathbb{N} \to \mathbb{N}$, public component length of the secret key $\mathsf{KES.pcl}: \mathbb{N} \to \mathbb{N}$, secret component length of the secret key $\mathsf{KES.scl}: \mathbb{N} \to \mathbb{N}$, message length $\mathsf{KES.ml}: \mathbb{N} \to \mathbb{N}$, signature length $\mathsf{KES.sigl}: \mathbb{N} \to \mathbb{N}$, and the maximum number of time periods $\mathsf{KES.T}: \mathbb{N} \to \mathbb{N}$. For $\lambda \in \mathbb{N}$ we require that any secret key $sk \in \{0,1\}^{\mathsf{KES.skl}(\lambda)}$ can be parsed as a pair (pc, sc) containing a public component $pc \in \{0,1\}^{\mathsf{KES.pcl}(\lambda)}$ and a secret component $sc \in \{0,1\}^{\mathsf{KES.scl}(\lambda)}$, such that $\mathsf{KES.skl}(\lambda) = \mathsf{KES.pcl}(\lambda) + \mathsf{KES.scl}(\lambda)$. Key generation algorithm $\mathsf{KES.Kg}$ takes 1^λ to return a public key $pk \in \{0,1\}^{\mathsf{KES.pkl}(1^\lambda)}$ and base (time period one) secret signing key $sk_1 \in \{0,1\}^{\mathsf{KES.skl}(\lambda)}$. Key update algorithm $\mathsf{KES.Up}$ takes $1^\lambda, pk, i$ and a secret key $sk_i \in \{0,1\}^{\mathsf{KES.skl}(\lambda)}$ for time period i to return a $\mathsf{KES.skl}(\lambda)$-bit se-

Game $\mathrm{FUFCL}^{\mathcal{A}}_{\mathsf{KES}}(\lambda)$	Game $\mathrm{FINDCL}^{\mathcal{A}}_{\mathsf{KEE}}(\lambda)$

$S \leftarrow \emptyset\,;\ t \leftarrow 1\,;\ t^* \leftarrow \mathsf{KES.T}(\lambda)+1$
$(pk, sk_1) \leftarrow\$\ \mathsf{KES.Kg}(1^\lambda)\,;\ (pc, sc) \leftarrow sk_1$
$(i, m, \sigma) \leftarrow\$\ \mathcal{A}^{\textsc{Up},\textsc{Lk},\textsc{Exp},\textsc{Sign}}(1^\lambda, pk, pc)$
$\mathsf{win}_1 \leftarrow (1 \le i < t^*) \wedge ((i, m, \sigma) \notin S)$
$\mathsf{win}_2 \leftarrow \mathsf{KES.Vf}(1^\lambda, pk, m, (i, \sigma))$
Return $(\mathsf{win}_1 \wedge \mathsf{win}_2)$
 	
$\underline{\textsc{Up}()}$
If $t < \mathsf{KES.T}(\lambda)$ then
  $sk_{t+1} \leftarrow\$\ \mathsf{KES.Up}(1^\lambda, pk, t, sk_t)$
  $(pc, sc) \leftarrow sk_{t+1}\,;\ t \leftarrow t+1$
  Return pc
Else return $\perp$
 	
$\underline{\textsc{Lk}(L)}$
$(pc, sc) \leftarrow sk_t\,;\ $ Return $L(sc)$
 	
$\underline{\textsc{Exp}()}$
$t^* \leftarrow t\,;\ $ Return sk_t
 	
$\underline{\textsc{Sign}(m)}$
$(t, \sigma) \leftarrow\$\ \mathsf{KES.Sig}(1^\lambda, pk, t, sk_t, m)$
$S \leftarrow S \cup \{(t, m, \sigma)\}\,;\ $ Return (t, σ)
	$b \leftarrow\$\ \{0,1\}\,;\ t \leftarrow 1\,;\ t^* \leftarrow \mathsf{KEE.T}(\lambda)+1$
	$(pk, sk_1) \leftarrow\$\ \mathsf{KEE.Kg}(1^\lambda)\,;\ (pc, sc) \leftarrow sk_1$
	$(i, m_0, m_1, state) \leftarrow\$\ \mathcal{A}_1^{\textsc{Up},\textsc{Lk},\textsc{Exp}}(1^\lambda, pk, pc)$
	If not $(1 \le i < t^*)$ then return false
	If $\lvert m_0 \rvert \ne \lvert m_1 \rvert$ then return false
	$(i, c) \leftarrow\$\ \mathsf{KEE.Enc}(1^\lambda, pk, i, m_b)$
	$b' \leftarrow\$\ \mathcal{A}_2(1^\lambda, state, (i, c))$
	Return $(b' = b)$
	
	$\underline{\textsc{Up}()}$
	If $t < \mathsf{KEE.T}(\lambda)$ then
	  $sk_{t+1} \leftarrow\$\ \mathsf{KEE.Up}(1^\lambda, pk, t, sk_t)$
	  $(pc, sc) \leftarrow sk_{t+1}\,;\ t \leftarrow t+1$
	  Return pc
	Else return $\perp$
	
	$\underline{\textsc{Lk}(L)}$
	$(pc, sc) \leftarrow sk_t\,;\ $ Return $L(sc)$
	
	$\underline{\textsc{Exp}()}$
	$t^* \leftarrow t\,;\ $ Return sk_t

Figure 2: Games defining forward unforgeability of key-evolving signature scheme KES under continual leakage, and forward indistinguishability of key-evolving encryption scheme KEE under continual leakage.

---

cret key for the next time period. Signing algorithm $\mathsf{KES.Sig}$ takes $1^\lambda, pk, i, sk_i$ and a message $m \in \{0,1\}^{\mathsf{KES.ml}(\lambda)}$ to return a pair (i, σ), where $\sigma \in \{0,1\}^{\mathsf{KES.sigl}(\lambda)}$ is a signature of m under secret key sk_i. Signature verification algorithm $\mathsf{KES.Vf}$ takes $1^\lambda, pk, m, (i, \sigma)$ to return a decision in $\{\mathsf{true}, \mathsf{false}\}$ regarding whether σ is a valid signature of message m relative to public key pk and time period $i \in [\mathsf{KES.T}(\lambda)]$. Correctness requires that $\mathsf{KES.Vf}(1^\lambda, pk, m, (i, \sigma)) = \mathsf{true}$ for all $\lambda \in \mathbb{N}$, all $m \in \{0,1\}^{\mathsf{KES.ml}(\lambda)}$, all $(pk, sk_1) \in [\mathsf{KES.Kg}(1^\lambda)]$, all $i \in [\mathsf{KES.T}(\lambda)]$, all $sk_2, \ldots, sk_i$ satisfying $sk_j \in [\mathsf{KES.Up}(1^\lambda, pk, j-1, sk_{j-1})]$ for $2 \le j \le i$, and all σ such that $(i, \sigma) \in [\mathsf{KES.Sig}(1^\lambda, pk, i, sk_i, m)]$.

<u>Forward unforgeability under continual leakage.</u> Consider game FUFCL of Fig. 2 associated to a key-evolving signature scheme KES and an adversary $\mathcal{A}$, where $\textsc{Lk}$ takes as input a Boolean circuit $L\colon \{0,1\}^{\mathsf{KES.scl}(\lambda)} \to \{0,1\}$. For $\lambda \in \mathbb{N}$ let $\mathsf{Adv}^{\mathsf{fufcl}}_{\mathsf{KES},\mathcal{A}}(\lambda) = \Pr[\mathrm{FUFCL}^{\mathcal{A}}_{\mathsf{KES}}(\lambda)]$. We say that FUFCL adversary $\mathcal{A}$ is *valid* if it makes at most one query to its $\textsc{Exp}$ oracle, and this is its last oracle query. We say that $\mathcal{A}$ is δ-bounded, where $\delta\colon \mathbb{N} \to [0,1]$, if $\mathcal{A}$ makes at most $\delta(\lambda) \cdot \mathsf{KES.scl}(\lambda)$ queries to $\textsc{Lk}$ *per time period.* That is, leakage on the secret component of secret key in any one time period is restricted to this number of bits. We say that KES is δ-FUFCL (δ-forward unforgeable under continual leakage) if $\mathsf{Adv}^{\mathsf{fufcl}}_{\mathsf{KES},\mathcal{A}}(\cdot)$ is negligible for all valid, δ-bounded PT adversaries $\mathcal{A}$.

The game begins by picking a public key pk and base secret key sk_1 for the first time period.

The adversary receives pk and the public component pc of the secret key sk_1. The current time period is t and the corresponding key, sk_t, is the one under attack. The SIGN oracle allows the adversary to obtain signatures under the current key. The adversary may obtain leakage about the secret component sc of the secret key sk_t via its leakage oracle LK. The latter takes an adversary-provided boolean circuit $L : \{0,1\}^{\mathsf{KES.scl}(\lambda)} \to \{0,1\}$ and returns $L(sc)$ as leakage. Note that the adversary is restricted to querying LK with circuits that output only one bit, but it may adaptively query the oracle multiple times to leak more bits. At any point the adversary may call UP to advance the key to the next stage, receiving as output the public component of the new secret key. Calls to SIGN, LK and UP may be adaptively interleaved. At any time the adversary also has the option of fully exposing the current secret key via its EXP oracle. The time period in which it does this is denoted t^*. At that point it is disallowed any further calls to its oracles and must terminate. To win it must output a valid message-signature pair relative to a time period prior to t^*, where valid means that the signature-verification algorithm $\mathsf{KES.Vf}$ accepts it, and that the message-signature pair for the particular time period was not previously received as an output of the SIGN oracle. Adversary's advantage is the probability that it wins.

Security under all keys is guaranteed as long as adversary learns at most a δ fraction of every secret key's secret component. If leakage exceeds this amount (modeled by an EXP query being made) then, rather than all being lost, forward security is provided, meaning security of prior keys is maintained. Forward-secure signatures as defined in [8] are the special case of FUFCL signatures for adversaries that make no LK queries. Signatures that are secure against continual leakage are the special case of FUFCL signatures for adversaries that make no EXP queries. Thus our model unifies the two notions under the new goal of forward unforgeability under continual leakage. Our definitions of key-evolving signatures and continual-leakage security are different from those used in the prior work [19, 14, 11, 12, 37, 27], but they are equivalent up to simple transformations, as explained below. The difference is that key-evolving signatures from the prior work are defined to use signing and verification algorithms that are oblivious to the current time period. However, a key-evolving scheme as per our definition can be constructed from a standard key-evolving scheme by using the latter to sign and verify messages of the form $i \,\|\, m$, which is a concatenation of the current time period i and a message m. Furthermore, a standard key-evolving scheme can be constructed from a key-evolving scheme as per our definition by building the secret keys of the standard scheme as $i \,\|\, sk_i$, containing the current time period i and the corresponding secret key sk_i for a scheme of our type. The resulting constructions of key-evolving signature schemes inherit the continual-leakage security of the original schemes.

KEY-EVOLVING ENCRYPTION SCHEMES. A key-evolving encryption scheme KEE specifies PT algorithms $\mathsf{KEE.Kg}$, $\mathsf{KEE.Up}$, $\mathsf{KEE.Enc}$ and $\mathsf{KEE.Dec}$, where $\mathsf{KEE.Dec}$ is deterministic. Associated to KEE are the following polynomials: secret-key length $\mathsf{KEE.skl} : \mathbb{N} \to \mathbb{N}$, public component length of the secret key $\mathsf{KEE.pcl} : \mathbb{N} \to \mathbb{N}$, secret component length of the secret key $\mathsf{KEE.scl} : \mathbb{N} \to \mathbb{N}$, message length $\mathsf{KEE.ml} : \mathbb{N} \to \mathbb{N}$, and the maximum number of time periods $\mathsf{KEE.T} : \mathbb{N} \to \mathbb{N}$. For $\lambda \in \mathbb{N}$ we require that any secret key $sk \in \{0,1\}^{\mathsf{KEE.skl}(\lambda)}$ can be parsed as a pair (pc, sc) containing a public component $pc \in \{0,1\}^{\mathsf{KEE.pcl}(\lambda)}$ and a secret component $sc \in \{0,1\}^{\mathsf{KEE.scl}(\lambda)}$, such that $\mathsf{KEE.skl}(\lambda) = \mathsf{KEE.pcl}(\lambda) + \mathsf{KEE.scl}(\lambda)$. Key generation algorithm $\mathsf{KEE.Kg}$ takes 1^λ to return a public key pk and base (time period one) secret signing key $sk_1 \in \{0,1\}^{\mathsf{KEE.skl}(\lambda)}$. Key update algorithm $\mathsf{KEE.Up}$ takes $1^\lambda, pk, i$ and a secret key $sk_i \in \{0,1\}^{\mathsf{KEE.skl}(\lambda)}$ for time period i to return a $\mathsf{KEE.skl}(\lambda)$-bit secret key for the next time period. Encryption algorithm $\mathsf{KEE.Enc}$ takes $1^\lambda, pk, i$ and a message $m \in \{0,1\}^{\mathsf{KEE.ml}(\lambda)}$ to return (i, c), where c is an encryption of m under pk for time period i. Decryption algorithm $\mathsf{KEE.Dec}$ takes $1^\lambda, pk, i, sk_i, (j, c)$ to return $m \in \{0,1\}^{\mathsf{KEE.ml}(\lambda)} \cup \{\bot\}$. Correctness requires that $\mathsf{KEE.Dec}(1^\lambda, pk, i, sk_i, (i, c)) = m$ for all $\lambda \in \mathbb{N}$, all $m \in \{0,1\}^{\mathsf{KEE.ml}(\lambda)}$, all

$(pk, sk_1) \in [\mathsf{KEE.Kg}(1^\lambda)]$, all $i \in [\mathsf{KEE.T}(\lambda)]$, all $sk_2, \ldots, sk_i$ satisfying $sk_j \in [\mathsf{KEE.Up}(1^\lambda, pk, j-1, sk_{j-1})]$ for $2 \le j \le i$, and all c such that $(i, c) \in [\mathsf{KEE.Enc}(1^\lambda, pk, i, m)]$.

Forward indistinguishability under continual leakage. Consider game FINDCL of Fig. 2 associated to a key-evolving encryption scheme KEE and an adversary $\mathcal{A}$, where Lk takes as input a Boolean circuit $L: \{0,1\}^{\mathsf{KEE.scl}(\lambda)} \to \{0,1\}$. For $\lambda \in \mathbb{N}$ let $\mathsf{Adv}^{\mathsf{findcl}}_{\mathsf{KEE},\mathcal{A}}(\lambda) = 2\Pr[\mathrm{FINDCL}^{\mathcal{A}}_{\mathsf{KEE}}(\lambda)] - 1$. We say that FINDCL adversary $\mathcal{A}$ is *valid* if it makes at most one query to its Exp oracle, and this is its last oracle query. We say that $\mathcal{A}$ is δ-bounded, where $\delta: \mathbb{N} \to [0,1]$, if $\mathcal{A}$ makes at most $\delta(\lambda) \cdot \mathsf{KEE.scl}(\lambda)$ queries to Lk *per time period*. We say that KEE is δ-FINDCL (δ-forward indistinguishable under continual leakage) if $\mathsf{Adv}^{\mathsf{findcl}}_{\mathsf{KEE},\mathcal{A}}(\cdot)$ is negligible for all valid, δ-bounded PT adversaries $\mathcal{A}$.

Game FINDCL is similar to game FUFCL in terms of allowing the adversary to obtain information about the secret keys in different time periods by providing it with oracle access to Up, Lk and Exp. Having finished making queries to its oracles, the adversary has to choose a time period (prior to the key exposure, if Exp was called) and a pair of challenge messages of equal length. The adversary then is given a challenge ciphertext for the specified time period, and it has to guess which of the two challenge messages was encrypted in order to win the game. Encryption secure against continual leakage as defined in [14] is the special case of FINDCL encryption for adversaries that make no Exp queries (these notions are equivalent up to simple transformations that are required due to different semantics across the definitions of key-evolving schemes, similar to the the case for key-evolving signature schemes that we discussed above). Forward-secure encryption as defined in [15] is the special case of FINDCL encryption for adversaries that make no Lk queries. Our model unifies the two under the new goal of forward indistinguishability under continual leakage.

Convention for adversary restrictions. Whenever we consider an adversary that meets certain conditions (e.g. is PT, valid, and δ-bounded), we require that it holds not just in the games defining security, but regardless of adversary's inputs and how its oracle queries are answered. This will help us to simplify the proof of Theorem 5.1 where an FUFCL adversary will be simulated in an environment that is different than the one it might expect from the FUFCL game.

4 FUFCL signatures from UFCL signatures

In this section we show how to construct a FUFCL signature scheme from any key-evolving signature scheme that is unforgeable under continual leakage (UFCL). The latter is a standard continual leakage security notion that is also a special case of FUFCL with respect to adversaries that do not query the Exp oracle. We define it below.

Unforgeability under continual leakage. Consider game FUFCL of Fig. 2 associated to a key-evolving signature scheme KES and an adversary $\mathcal{A}$, where Lk takes as input a Boolean circuit $L: \{0,1\}^{\mathsf{KES.scl}(\lambda)} \to \{0,1\}$. For $\lambda \in \mathbb{N}$ let $\mathsf{Adv}^{\mathsf{ufcl}}_{\mathsf{KES},\mathcal{A}}(\lambda) = \Pr[\mathrm{FUFCL}^{\mathcal{A}}_{\mathsf{KES}}(\lambda)]$. We say that UFCL adversary $\mathcal{A}$ is valid if it makes *no queries* to its Exp oracle. We say that $\mathcal{A}$ is δ-bounded, where $\delta: \mathbb{N} \to [0,1]$, if $\mathcal{A}$ makes at most $\delta(\lambda) \cdot \mathsf{KES.scl}(\lambda)$ queries to Lk *per time period*. We say that KES is δ-UFCL (δ-unforgeable under continual leakage) if $\mathsf{Adv}^{\mathsf{ufcl}}_{\mathsf{KES},\mathcal{A}}(\cdot)$ is negligible for all valid, δ-bounded PT adversaries $\mathcal{A}$.

FUFCL signatures from a binary tree of UFCL signatures. We use the binary tree construction of forward secure signatures by Bellare and Miner [8], with any continual leakage secure signature scheme as the base scheme. The construction is also similar to the one by Faust el al. [23], but the goal they achieve is different.

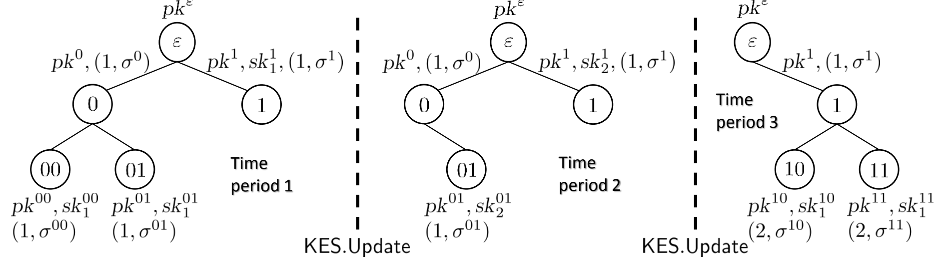

Figure 3: The construction of a key-evolving signature scheme KES from depth-2 binary tree, showing the information stored in the tree during the first 3 out of the 4 possible time periods. Each node corresponds to an independent instance of the underlying key-evolving signature scheme SIG. Superscripts denote the positions of displayed entities in the tree, and subscripts denote the time periods of individual secret keys.

We now describe the high-level idea of our construction. Let SIG be a key-evolving signature scheme; we compose many SIG key-pairs into a binary tree to build a new key-evolving signature scheme KES. We will then show that if SIG is UFCL then KES is FUFCL. Fig. 3 shows an example of a binary tree of depth 2 for multiple subsequent time periods.

Each node of the tree containts an independently generated SIG public key. A node may also contain the corresponding secret key, but the secret key is erased as soon as both of its child nodes have been generated; this will ensure that the constructed KES scheme is forward-secure. Each non-root node contains a signature of its public key under its parent node's secret key.

The SIG public key of the root node is used as the public key of the KES scheme. The leaf nodes of the tree are used to produce KES signatures, each for a separate time period, meaning that a tree-based construction of depth h has 2^h time periods. The secret key of the KES scheme contains all information about the current tree structure besides its root's public key; the secret component of a KES secret key contains all currently available SIG secret keys, whereas everything else is stored inside its public component. The KES signature of a message m includes a SIG signature of m for a secret key of (the current) leaf node, along with information about the path from the root node to this leaf; for each non-root node on this path, it includes this node's public key and its signature under its parent's secret key. This allows to verify signatures having only the public key of the root node.

The key update procedure of the KES scheme modifies the tree to generate the next leftmost leaf node (if necessary) and set it as the one that is used to generate signatures. For this to be possible, the current tree structure always contains the nodes that branch to the right of those nodes that lie on the path from the root to the current leaf node. To ensure the forward security of KES, the old leaf node is erased at the end of the update procedure. For the continual-leakage security of KES, all other SIG secret keys are updated at the end of the update procedure, meaning their individual time periods get increased (this allows to repeatedly leak on them during each separate KES time period).

For example, consider the tree-based key-evolving scheme KES from Fig. 3 in time period 1. Its public key is pk^ε. Its secret key $sk = (pc, sc)$ consists of a secret component $sc = (sk_1^{00}, sk_1^{01}, sk_1^1)$ and a public component $pc = ((pk^0, 1, \sigma^0), (pk^{00}, 1, \sigma^{00}), (pk^{01}, 1, \sigma^{01}), (pk^1, 1, \sigma^1))$. A signature of message m is $(1, \sigma)$ for $\sigma = ((pk^0, 1, \sigma^0), (pk^{00}, 1, \sigma^{00}), (1, \sigma'))$ and $\sigma' \leftarrow\$ \mathsf{SIG.Sig}(1^\lambda, pk^{00}, 1, sk_1^{00}, m)$. The first component of $(1, \sigma)$ denotes the time period of KES, whereas the first component of $(1, \sigma')$ denotes the time period of key sk_1^{00}. The node information of the form $(pk^0, 1, \sigma^0)$

indicates that σ^0 is a signature of pk^0 under its parent node's secret key, and the latter was used when its time period was 1.

<u>Our construction and its security.</u> We define a key-evolving signature scheme $\mathsf{KES} = \mathsf{KES\text{-}TREE}[\mathsf{SIG}, h]$ as a binary-tree construction described above, where each node of the binary tree contains a key pair of another key-evolving signature scheme SIG, and the height of the binary tree is defined by a polynomial h (parameterized by a security parameter). The formal definition of KES-TREE is in Appendix A. This construction is straightforward, but also very detailed.

We show that if SIG is UFCL-secure, then KES is FUFCL-secure. Note that for any security parameter $\lambda \in \mathbb{N}$, the secret key of KES contains $h(\lambda) + 1$ secret keys of SIG. Therefore, the continual-leakage fraction supported by KES is $h(\lambda) + 1$ times worse than that of SIG.

Theorem 4.1 *Let* $\delta : \mathbb{N} \to [0, 1]$*. Let* SIG *be a* δ*-UFCL key-evolving signature scheme with* $\mathsf{SIG.ml} = \mathsf{SIG.pkl} + 1$*. Let* $h : \mathbb{N} \to \mathbb{N}$ *a polynomial such that* $2^{h(\lambda)-1} \leq \mathsf{SIG.T}(\lambda)$ *for all* $\lambda \in \mathbb{N}$*. Let* $\gamma(\lambda) = \delta(\lambda)/(h(\lambda) + 1)$ *for all* $\lambda \in \mathbb{N}$*. Then the key-evolving signature scheme* $\mathsf{KES} = \mathsf{KES\text{-}TREE}[\mathsf{SIG}, h]$ *is* γ*-FUFCL.*

The proof is in Appendix B. Informally, it proceeds as follows. Assume that a PT adversary $\mathcal{A}$ breaks the FUFCL-security of KES. In order to do that, it has to forge a valid message-signature pair of KES scheme for some time period i (which must be prior to the time period of full key exposure). In terms of the underlying binary tree structure, this means that $\mathcal{A}$ succesfully forges a valid message-signature pair for one of the SIG verification keys that lie on the path from the root of the KES binary tree to the leaf node that is associated to time period i. We build a PT adversary $\mathcal{B}$ against the UFCL-security of SIG as follows. It attempts to guess the KES binary tree node x that will be attacked by $\mathcal{A}$ (out of the $2^{h(\lambda)+1} - 1$ possible nodes); this node will correspond to the challenge key-pair in game UFCL. It then generates SIG key pairs for all other $2^{h(\lambda)+1} - 2$ nodes, and uses its UFCL security game oracles to answer any of $\mathcal{A}$'s oracle queries that depend on the secret key of node x (which is unknown to $\mathcal{B}$). If $\mathcal{B}$ guessed the position of the challenge node correctly, then $\mathcal{A}$ breaking the FUFCL-security of KES results in $\mathcal{B}$ breaking the UFCL-security of SIG.

<u>Extensions.</u> Note that a binary tree pre-order traveral can be used to associate *each* node of the binary tree with a separate time period of the resulting signature scheme, rather than only use the leaf nodes as we currently do. This was done in some of the previous results that used tree-based construction, such as [15, 23].

5 A unified paradigm for constructing FS+CL schemes

In this work we propose to build primitives that are secure against continual leakage and simultaneously forward secure, the latter serving as a second line of defense. Towards that end, in Section 3 we defined key-evolving signature and encryption schemes, and the corresponding security notions. An important part of both schemes is a key update procedure that allows to repeatedly evolve a secret key in the presence of a single, fixed public key. In this section we define *key-evolution schemes* that model this process. We consider a security notion called *forward one-wayness under continual leakage* (FOWCL) that formalizes properties of the key evolution that are necessary (although not sufficient) for the security of both signature and encryption schemes as defined in Section 3. We discuss how to achieve FOWCL key-evolution schemes, and we show how to use them to build key-evolving signature and encryption schemes that inherit the continual-leakage fraction of the former in a modular way.

$$
\begin{array}{l}
\underline{\text{Game FOWCL}^{\mathcal{A}}_{\mathsf{KE}}(\lambda)} \\
t \leftarrow 1\, ;\ t^* \leftarrow \mathsf{KE.T}(\lambda)+1 \\
(pk, sk_1) \leftarrow\!\!\$\ \mathsf{KE.Kg}(1^\lambda)\, ;\ (pc, sc) \leftarrow sk_1 \\
(i, sk) \leftarrow\!\!\$\ \mathcal{A}^{\textsc{Up},\textsc{Lk},\textsc{Exp}}(1^\lambda, pk, pc) \\
\mathsf{win}_1 \leftarrow (1 \le i < t^*) \\
\mathsf{win}_2 \leftarrow \mathsf{KE.Vf}(1^\lambda, pk, i, sk) \\
\text{Return } (\mathsf{win}_1 \wedge \mathsf{win}_2) \\
\\
\underline{\textsc{Up}()} \\
\text{If } t < \mathsf{KE.T}(\lambda) \text{ then} \\
\quad sk_{t+1} \leftarrow\!\!\$\ \mathsf{KE.Up}(1^\lambda, pk, t, sk_t) \\
\quad (pc, sc) \leftarrow sk_{t+1}\, ;\ t \leftarrow t+1 \\
\quad \text{Return } pc \\
\text{Else return } \bot \\
\\
\underline{\textsc{Lk}(L)} \\
(pc, sc) \leftarrow sk_t\, ;\ \text{Return } L(sc) \\
\\
\underline{\textsc{Exp}()} \\
t^* \leftarrow t\, ;\ \text{Return } sk_t
\end{array}
$$

Figure 4: Game defining forward one-wayness of key-evolution scheme KE under continual leakage.

KEY-EVOLUTION SCHEMES. A key-evolution scheme KE specifies PT algorithms $\mathsf{KE.Kg}$, $\mathsf{KE.Up}$ and $\mathsf{KE.Vf}$, where $\mathsf{KE.Vf}$ is deterministic. Associated to KE are the following polynomials: secret-key length $\mathsf{KE.skl} : \mathbb{N} \to \mathbb{N}$, public component length of the secret key $\mathsf{KE.pcl} : \mathbb{N} \to \mathbb{N}$, secret component length of the secret key $\mathsf{KE.scl} : \mathbb{N} \to \mathbb{N}$, and the maximum number of time periods $\mathsf{KE.T} : \mathbb{N} \to \mathbb{N}$. For $\lambda \in \mathbb{N}$ we require that any secret key $sk \in \{0,1\}^{\mathsf{KE.skl}(\lambda)}$ can be parsed as a pair (pc, sc) containing a public component $pc \in \{0,1\}^{\mathsf{KE.pcl}(\lambda)}$ and a secret component $sc \in \{0,1\}^{\mathsf{KE.scl}(\lambda)}$, such that $\mathsf{KE.skl}(\lambda) = \mathsf{KE.pcl}(\lambda) + \mathsf{KE.scl}(\lambda)$. Key generation algorithm $\mathsf{KE.Kg}$ takes 1^λ to return a public key pk and base (time period one) secret key $sk_1 \in \{0,1\}^{\mathsf{KE.skl}(\lambda)}$. Key update algorithm $\mathsf{KE.Up}$ takes $1^\lambda, pk, i$ and a secret key $sk_i \in \{0,1\}^{\mathsf{KE.skl}(\lambda)}$ for time period i to return a $\mathsf{KE.skl}(\lambda)$-bit secret key for the next time period. Key verification algorithm $\mathsf{KE.Vf}$ takes $1^\lambda, pk, i, sk_i$ to return a decision in $\{\mathsf{true}, \mathsf{false}\}$ regarding whether sk_i is a valid secret key relative to public key pk and time period $i \in [\mathsf{KE.T}(\lambda)]$. Correctness requires that $\mathsf{KE.Vf}(1^\lambda, pk, i, sk_i) = \mathsf{true}$ for all $\lambda \in \mathbb{N}$, all $(pk, sk_1) \in [\mathsf{KE.Kg}(1^\lambda)]$, all $i \in [\mathsf{KE.T}(\lambda)]$ and all $sk_2, \ldots, sk_i$ satisfying $sk_j \in [\mathsf{KE.Up}(1^\lambda, pk, j-1, sk_{j-1})]$ for $2 \le j \le i$. That is, all secret keys that can be obtained via correct updates starting from sk_1 should pass the verification test.

FORWARD SECURITY UNDER CONTINUAL LEAKAGE. Consider game FOWCL of Fig. 4 associated to a key-evolution scheme KE and an adversary $\mathcal{A}$, where LK takes a Boolean circuit $L : \{0,1\}^{\mathsf{KE.scl}(\lambda)} \to \{0,1\}$. For $\lambda \in \mathbb{N}$ let $\mathsf{Adv}^{\mathsf{fowcl}}_{\mathsf{KE},\mathcal{A}}(\lambda) = \Pr[\text{FOWCL}^{\mathcal{A}}_{\mathsf{KE}}(\lambda)]$. We say that FOWCL adversary $\mathcal{A}$ is *valid* if it makes at most one query to its EXP oracle, and this is its last oracle query. We say that $\mathcal{A}$ is δ-bounded, where $\delta : \mathbb{N} \to [0,1]$, if $\mathcal{A}$ makes at most $\delta(\lambda) \cdot \mathsf{KE.scl}(\lambda)$ queries to LK *per time period*. We say that KE is δ-FOWCL (δ-forward one-way under continual leakage) if $\mathsf{Adv}^{\mathsf{fowcl}}_{\mathsf{KE},\mathcal{A}}(\cdot)$ is negligible for all valid, δ-bounded PT adversaries $\mathcal{A}$.

Game FOWCL is similar to games FINDCL and FUFCL from Section 3 in terms of allowing

the adversary to obtain information about the secret keys in different time periods by providing it with oracle access to UP, LK and EXP. In order to win, the adversary must output a valid secret key relative to a time period prior to t^* (when exposure happened), where valid means that the key-verification function of KE accepts it. Adversary's advantage is the probability that it wins. To recover relations that are one-way against continual leakage as defined in [19], one could consider adversaries that make no EXP queries. The two notions are equivalent up to simple transformations that are required due to different semantics between the definitions of key-evolving schemes (similar to the the case of key-evolving signature schemes as discussed in Section 3). Considering adversaries that make no LK queries captures relations that provide forward one wayness. Our model unifies the two security notions.

The familiar requirement for security of a key is that it be indistinguishable from random. This is not achievable when the adversary is in possession of leakage on the key. The requirement we make, following [19], is very weak, namely that the adversary be unable to fully recover a valid key (one-wayness). Then the difficulty is to be able to use such a key for a cryptographic application. This will be done via witness primitives – encryption and signatures.

WITNESS ENCRYPTION AND WITNESS SIGNATURES. Witness encryption [26, 6, 28] for an **NP**-relation R allows anyone to encrypt messages with respect to any instance $x \in \{0,1\}^*$. In order to decrypt a message encrypted to x, it is necessary to know a witness w such that $\mathsf{R.Vf}(x, w) = \mathsf{true}$. Witness signatures [16, 7] for an **NP**-relation R allow to sign messages with respect to an instance-witness pair (x, w) such that $\mathsf{R.Vf}(x, w) = \mathsf{true}$. In order to verify a signature produced this way, it is sufficient to know the instance x that was used in the signing process. In the next sections we will discuss both primitives in more details.

COMPOSING KEY-EVOLUTION SCHEMES WITH WITNESS PRIMITIVES. We now discuss how to use an arbitrary FOWCL key-evolution scheme in order to obtain a FUFCL signature scheme and a FINDCL encryption scheme. This is done in a generic way via a unified paradigm. Namely in Sections 5.1 and 5.2 we show that FOWCL + Witness-X yields FS+CL-secure X for X=signatures and X=encryption.

Let us explain the issues and the idea. The FOWCL key-evolution scheme provides a way to obtain keys that remain unrecoverable in the FS+CL sense. But it is not clear how to use these keys for signatures or encryption. The reason is that signature and encryption schemes usually require keys of very specific structure that varies from scheme to scheme, but here we are handed keys of a complex structure that are not obviously suitable for any particular application. But witness primitives are, in the terminology of [7], highly "key-versatile". That is, they are able to provide security of the application assuming nothing more than that secret keys are hard to recover from the public key. We will combine them with key evolution to achieve signatures and encryption. Note that our security proofs will require some form of extractability from both witness primitives. Furthermore, we know no direct constructions of FOWCL key-evolution schemes, but any FUFCL schemes we build based on the construction in Section 4 are also FOWCL by definition.

The encryption and signature schemes constructed using our approach will inherit the leakage rate of the used key-evolution scheme, as opposed to the direct construction in Section 4 where the leakage rate detoriates logarithmically with the maximum number of time periods. Another advantage of this approach is modularity. We do not need to re-enter any details of our construction of a FOWCL key-evolution scheme, leading to conceptual simplicity. Also, should any new, more efficient or better constructions of FOWCL key-evolution schemes arise in the future, the transforms in this section can be invoked to automatically turn them into FS+CL signature and encryption schemes.

Game $\mathrm{SIM}^{\mathcal{A}}_{\mathsf{WS},\mathsf{R}}(\lambda)$

$b \leftarrow\$ \{0,1\}$
$wp_1 \leftarrow\$ \mathsf{WS.Pg}(1^\lambda)$
$(wp_0, std, xtd) \leftarrow\$ \mathsf{WS.SimPg}(1^\lambda)$
$b' \leftarrow\$ \mathcal{A}^{\mathrm{SIGN}}(1^\lambda, wp_b)$
Return $(b' = b)$

$\mathrm{SIGN}(x, w, m)$

If not $\mathsf{R.Vf}(x, w)$ then return $\perp$
If $b = 1$ then $\sigma \leftarrow\$ \mathsf{WS.Sig}(1^\lambda, wp_1, x, w, m)$
Else $\sigma \leftarrow\$ \mathsf{WS.SimSig}(1^\lambda, wp_0, x, std, m)$
Return σ

Game $\mathrm{EXT}^{\mathcal{A}}_{\mathsf{WS},\mathsf{R}}(\lambda)$

$Q \leftarrow \emptyset$; $(wp, std, xtd) \leftarrow\$ \mathsf{WS.SimPg}(1^\lambda)$
$(x, m, \sigma) \leftarrow\$ \mathcal{A}^{\mathrm{SIGN}}(1^\lambda, wp)$
If $(x, m, \sigma) \in Q$ then return false
If $x \notin \mathcal{L}(\mathsf{R})$ then return false
$d \leftarrow \mathsf{WS.Vf}(1^\lambda, wp, x, m, \sigma)$
If not d then return false
$w \leftarrow\$ \mathsf{WS.Ext}(1^\lambda, wp, x, xtd, m, \sigma)$
Return not $\mathsf{R.Vf}(x, w)$

$\mathrm{SIGN}(x, w, m)$

If not $\mathsf{R.Vf}(x, w)$ then return $\perp$
$\sigma \leftarrow\$ \mathsf{WS.SimSig}(1^\lambda, wp, x, std, m)$
$Q \leftarrow Q \cup \{(x, m, \sigma)\}$; Return σ

Figure 5: Games defining signature simulatability of witness signature scheme WS for **NP**-relation R, and witness extractability of witness signature scheme WS for **NP**-relation R.

5.1 FUFCL signatures from FOWCL key-evolution schemes

In this section we define witness signatures and show how to use them in order to transform a FOWCL key-evolution scheme into a FUFCL signature scheme.

Witness signatures. Let R be an **NP**-relation as defined in Section 2. A witness signature scheme WS for R specifies PT algorithms WS.Pg, WS.Sig, WS.Vf, WS.SimPg, WS.SimSig and WS.Ext, where WS.Vf is deterministic. Associated to WS is a message length polynomial $\mathsf{WS.ml}: \mathbb{N} \to \mathbb{N}$. Parameter generation algorithm WS.Pg takes 1^λ to return public parameters wp. Signing algorithm WS.Sig takes 1^λ, wp, an instance $x \in \{0,1\}^*$, a witness $w \in \{0,1\}^*$ and a message $m \in \{0,1\}^{\mathsf{WS.ml}(\lambda)}$ to return a signature σ. Signature verification algorithm WS.Vf takes $1^\lambda, wp, x, m, \sigma$ to return a decision in $\{\mathsf{true}, \mathsf{false}\}$. Correctness requires that $\mathsf{WS.Vf}(1^\lambda, wp, x, m, \sigma) = \mathsf{true}$ for all $\lambda \in \mathbb{N}$, all $wp \in [\mathsf{WS.Pg}(1^\lambda)]$, all x, w such that $\mathsf{R.Vf}(x, w) = \mathsf{true}$, all $m \in \{0,1\}^{\mathsf{WS.ml}(\lambda)}$ and all $\sigma \in [\mathsf{WS.Sig}(1^\lambda, wp, x, w, m)]$.

Simulated parameter generation algorithm WS.SimPg takes 1^λ to return simulated parameters wp, a signing trapdoor std and an extraction trapdoor xtd. Simulated signing algorithm WS.SimSig takes $1^\lambda, wp$, an instance x, signing trapdoor std and a message m (but no witness) to return a simulated signature σ. Extraction algorithm WS.Ext takes 1^λ, wp, instance x, extraction trapdoor xtd, message m and signature σ to return a candidate witness w for x.

Signature simulatability. Consider game SIM of Fig. 5 associated to an **NP**-relation R, a witness signature scheme WS for R, and an adversary $\mathcal{A}$. For $\lambda \in \mathbb{N}$ let $\mathsf{Adv}^{\mathsf{sim}}_{\mathsf{WS},\mathsf{R},\mathcal{A}}(\lambda) = 2\Pr[\mathrm{SIM}^{\mathcal{A}}_{\mathsf{WS},\mathsf{R}}(\lambda)] - 1$. We say that WS, R is signature simulatable if $\mathsf{Adv}^{\mathsf{sim}}_{\mathsf{WS},\mathsf{R},\mathcal{A}}(\cdot)$ is negligible for every PT adversary $\mathcal{A}$. This requires that the signature simulator, given simulated auxiliary parameters and a signature trapdoor, can produce a signature σ indistinguishable from the real one produced under the witness, when not just the message m, but even the instance x and witness w, are adaptively chosen by the adversary.

Witness extractability. Consider game EXT of Fig. 5 associated to an **NP**-relation R, a witness signature scheme WS for R, and an adversary $\mathcal{A}$. For $\lambda \in \mathbb{N}$ let $\mathsf{Adv}^{\mathsf{ext}}_{\mathsf{WS},\mathsf{R},\mathcal{A}}(\lambda) = \Pr[\mathrm{EXT}^{\mathcal{A}}_{\mathsf{WS},\mathsf{R}}(\lambda)]$.

$\underline{\mathsf{R.Vf}(x, sk)}$
$(1^\lambda, pk, i) \leftarrow x$
$\mathsf{win}_1 \leftarrow (1 \le i \le \mathsf{KE.T}(\lambda))$
$\mathsf{win}_2 \leftarrow (\mathsf{KE.Vf}(1^\lambda, pk, i, sk))$
Return $(\mathsf{win}_1 \wedge \mathsf{win}_2)$

Figure 6: **NP**-relation KE-REL = KE-REL[KE].

We say that WS, R is witness extractable if $\mathsf{Adv}^{\mathsf{ext}}_{\mathsf{WS},\mathsf{R},\mathcal{A}}(\cdot)$ is negligible for every PT adversary $\mathcal{A}$. This requires that the witness extractor, given simulated auxiliary parameters and an extraction trapdoor, can extract from any valid forgery relative to x an underlying witness w, even when x is chosen by the adversary and the adversary can adaptively obtain simulated signatures under instances and witnesses of its choice.

Obtaining witness signatures. Witness signatures as we define them above are effectively another name for Signatures of Knowledge as defined by Chase and Lysyanskaya [16] and refined by Bellare, Meiklejohn and Thomson [7]. Indeed the latter say that one might refer to this primitive as witness signatures, and we have followed that naming suggestion in order to have a unified terminology across encryption and signatures. The construction uses simulation sound extractable (SSE) NIZKs and follows [16, 20, 7]. Given any **NP**-relation R and polynomial p, it is possible to construct a witness signature scheme WS such that WS, R are signature simulatable and witness extractable and also $\mathsf{WS.ml} = p$. We omit the details and assume this capability in what follows. Note that these witness signatures are different from the ones of Goyal, Jain and Khurana [29]. In the latter, public parameters are not allowed and they show that in this case witness signatures are impossible. In our case, the public parameters can simply be put into the public key of the scheme we are constructing and are not an added assumption. In this case, as noted, witness signatures are easily constructed from NIZKs.

Construction of a FUFCL signature scheme. Assume we are given a key-evolution scheme KE that is FOWCL secure as defined in Section 5. We want to build a FUFCL secure key-evolving signature scheme. The difficulty is that the keys in KE may not have the structure required for any particular signature scheme and furthermore the security guarantee on them is weak, namely just that they are hard to recover in full. We achieve our ends through witness signatures. Informally, we associate to KE an **NP**-relation in which the role of the instance x is played by a triple $(1^\lambda, pk, i)$ containing security parameter, public key and time period for KE, and the role of the witness w is played by a secret key sk for KE. We then define a key-evolving signature scheme that uses this **NP**-relation to produce and verify signatures, as shown below.

NP-relation KE-REL. Let KE be a key-evolution scheme. We build an **NP**-relation R = KE-REL[KE] as defined in Fig. 6, where R.wl = KE.skl.

Key-evolving signature scheme WITNESS-KES. Let KE be a key-evolution scheme. Let WS be a witness signature scheme for the **NP**-relation R = KE-REL[KE]. We build a key-evolving signature scheme KES = WITNESS-KES[KE, WS] as defined in Fig. 7. The values of KES.skl, KES.pcl, KES.scl, KES.T are same as those of KE, and the values of KES.ml, KES.sigl are inherited from WS.

We now show that if KE is FOWCL-secure and WS is signature simulatable and witness extractable with respect to R, then KES is FUFCL-secure.

Theorem 5.1 *Let $\delta : \mathbb{N} \to [0, 1]$. Let KE be a δ-FOWCL key-evolution scheme. Let R = KE-REL[KE] be the **NP**-relation as defined above. Let WS be a witness signature scheme for R. Assume*

$\underline{\mathsf{KES.Kg}(1^\lambda)}$	$\underline{\mathsf{KES.Sig}(1^\lambda, (pk, wp), i, sk_i, m)}$
$(pk, sk_1) \leftarrow\$ \mathsf{KE.Kg}(1^\lambda)$	$\sigma \leftarrow\$ \mathsf{WS.Sig}(1^\lambda, wp, (1^\lambda, pk, i), sk_i, m)$
$wp \leftarrow\$ \mathsf{WS.Pg}(1^\lambda)$	Return (i, σ)
Return $((pk, wp), sk_1)$	$\underline{\mathsf{KES.Vf}(1^\lambda, (pk, wp), m, (i, \sigma))}$
$\underline{\mathsf{KES.Up}(1^\lambda, (pk, wp), i, sk_i)}$	$d_1 \leftarrow (1 \le i \le \mathsf{KE.T}(\lambda))$
$sk_{i+1} \leftarrow\$ \mathsf{KE.Up}(1^\lambda, pk, i, sk_i)$	$d_2 \leftarrow \mathsf{WS.Vf}(1^\lambda, wp, (1^\lambda, pk, i), m, \sigma)$
Return sk_{i+1}	Return $(d_1 \wedge d_2)$

Figure 7: Key-evolving signature scheme $\mathsf{KES} = \mathsf{WITNESS\text{-}KES}[\mathsf{KE}, \mathsf{WS}]$.

WS, R *is signature simulatable and witness extractable. Then key-evolving signature scheme* $\mathsf{KES} = \mathsf{WITNESS\text{-}KES}[\mathsf{KE}, \mathsf{WS}]$ *is δ-FUFCL secure.*

The proof idea is as follows. In structure, it follows proofs from [7]. Given an adversary $\mathcal{A}$ attacking FUFCL security of KES we want to build an adversary $\mathcal{B}$ attacking FOWCL security of KE. Adversary $\mathcal{B}$ can trivially answer UP, LK, EXP queries of $\mathcal{A}$ via its own corresponding oracles, but faces two problems. The first is to answer SIGN queries of $\mathcal{A}$ without having any of the secret keys. The second is that, even if $\mathcal{A}$ can be run to completion, what it returns is a forgery, and what $\mathcal{B}$ needs to win its game is a valid secret key. To solve these problems, $\mathcal{B}$ will generate simulated parameters for WS along with the associated trapdoors, answer SIGN queries of $\mathcal{A}$ via simulated signatures, and then use the witness extractor to extract a secret key from the forgery.

To show that this works we use a game sequence. First we use the signature simulatability of WS to switch from game FUFCL to one with simulated parameters and signatures. Next we use witness extractability of WS to show that whenever $\mathcal{A}$ breaks the FUFCL security of KES, one can use the provided forgery to succesfully extract a valid witness for the used relation. Since a valid witness in our relation consists of the secret key for KE, this allows us to build adversary $\mathcal{B}$ that simulates $\mathcal{A}$ and wins the game against FOWCL security of KE by returning a witness it extracted from $\mathcal{A}$'s forgery.

Proof of Theorem 5.1: Let $\mathcal{A}$ be a valid, δ-bounded PT adversary attacking KES in game FUFCL. We build a valid, δ-bounded PT adversary $\mathcal{B}$ attacking KE in game FOWCL, and PT adversaries $\mathcal{A}_1, \mathcal{A}_2$ attacking signature simulatability and witness extractability of WS, R giving

$$\mathsf{Adv}^{\mathsf{fufcl}}_{\mathsf{KES},\mathcal{A}}(\lambda) \le \mathsf{Adv}^{\mathsf{sim}}_{\mathsf{WS},\mathsf{R},\mathcal{A}_1}(\lambda) + \mathsf{Adv}^{\mathsf{ext}}_{\mathsf{WS},\mathsf{R},\mathcal{A}_2}(\lambda) + \mathsf{Adv}^{\mathsf{fowcl}}_{\mathsf{KE},\mathcal{B}}(\lambda)$$

for all $\lambda \in \mathbb{N}$. This justifies the claim in the theorem statement.

Consider games G_0, G_1, G_2 of Fig. 8. Lines not annotated with comments are common to all games. Game G_0 is equivalent to game $\mathrm{FUFCL}^{\mathcal{A}}_{\mathsf{KES}}(\lambda)$ with the code of KES expanded according to its definition, so $\Pr[\mathrm{G}_0] = \Pr[\mathrm{FUFCL}^{\mathcal{A}}_{\mathsf{KES}}(\lambda)]$. Game G_1 switches to using simulated parameters and signatures. Game G_2 additionally requires that the forgery produced by $\mathcal{A}$ allows to extract a valid secret key for the corresponding time period. We build PT adversaries $\mathcal{A}_1, \mathcal{A}_2, \mathcal{B}$ so that for all $\lambda \in \mathbb{N}$,

$$\Pr[\mathrm{G}_0] - \Pr[\mathrm{G}_1] = \mathsf{Adv}^{\mathsf{sim}}_{\mathsf{WS},\mathsf{R},\mathcal{A}_1}(\lambda) \ , \tag{1}$$

$$\Pr[\mathrm{G}_1 \text{ sets } \mathsf{bad}] \le \mathsf{Adv}^{\mathsf{ext}}_{\mathsf{WS},\mathsf{R},\mathcal{A}_2}(\lambda) \ , \tag{2}$$

$$\Pr[\mathrm{G}_2] \le \mathsf{Adv}^{\mathsf{fowcl}}_{\mathsf{KE},\mathcal{B}}(\lambda) \ . \tag{3}$$

Games G_1 and G_2 are identical until bad, so by the Fundamental Lemma of Game-Playing [9] and

```
Games G_0–G_2
S ← ∅ ; t ← 1 ; t* ← KE.T(λ) + 1
(pk, sk_1) ←$ KE.Kg(1^λ) ; (pc, sc) ← sk_1
wp ←$ WS.Pg(1^λ)                                   // G_0
(wp, std, xtd) ←$ WS.SimPg(1^λ)                    // G_1, G_2
(i, m, σ) ←$ A^{Up,Lk,Exp,Sign}(1^λ, (pk, wp), pc)
sk ←$ WS.Ext(1^λ, wp, (1^λ, pk, i), xtd, m, σ)
win_1 ← (1 ≤ i < t*) ∧ ((i, m, σ) ∉ S)
win_2 ← WS.Vf(1^λ, wp, (1^λ, pk, i), m, σ)
d ← false
If (win_1 ∧ win_2) then
   d ← true
   If not KE.Vf(1^λ, pk, i, sk) then
      bad ← true
      d ← false                                    // G_2
Return d

Up()
If t < KE.T(λ) then
   sk_{t+1} ←$ KE.Up(1^λ, pk, t, sk_t)
   (pc, sc) ← sk_{t+1} ; t ← t + 1
   Return pc
Else return ⊥

Lk(L)
(pc, sc) ← sk_t ; Return L(sc)

Exp()
t* ← t ; Return sk_t

Sign(m)
σ ←$ WS.Sig(1^λ, wp, (1^λ, pk, t), sk_t, m)        // G_0
σ ←$ WS.SimSig(1^λ, wp, (1^λ, pk, t), std, m)      // G_1, G_2
S ← S ∪ {(t, m, σ)} ; Return (t, σ)
```

Figure 8: Games G_0–G_2 for proof of Theorem 5.1.

---

the above, for all $\lambda \in \mathbb{N}$ we have

$$\begin{aligned}
&\mathsf{Adv}^{\mathsf{fufcl}}_{\mathsf{KES},\mathcal{A}}(\lambda) = \Pr[\mathrm{FUFCL}^{\mathcal{A}}_{\mathsf{KES}}(\lambda)] = \Pr[\mathrm{G}_0] \\
&= (\Pr[\mathrm{G}_0] - \Pr[\mathrm{G}_1]) + (\Pr[\mathrm{G}_1] - \Pr[\mathrm{G}_2]) + \Pr[\mathrm{G}_2] \\
&\le \mathsf{Adv}^{\mathsf{sim}}_{\mathsf{WS},\mathsf{R},\mathcal{A}_1}(\lambda) + \mathsf{Adv}^{\mathsf{ext}}_{\mathsf{WS},\mathsf{R},\mathcal{A}_2}(\lambda) + \mathsf{Adv}^{\mathsf{fowcl}}_{\mathsf{KE},\mathcal{B}}(\lambda).
\end{aligned}$$

This bounds the advantage of $\mathcal{A}$ as required for the theorem statement. We now provide the constructions of $\mathcal{A}_1, \mathcal{A}_2, \mathcal{B}$.

Consdier PT adversaries $\mathcal{A}_1, \mathcal{A}_2$ as defined in Fig. 9. The procedures to simulate the oracles of $\mathcal{A}$ are the same for both and thus for brevity written only once.

Adversaries $\mathcal{A}_1, \mathcal{A}_2$ generate their own keys for KE and can thus simulate the UP, LK, EXP oracles of $\mathcal{A}$ directly. They simulate $\mathcal{A}$'s SIGN oracle via their own SIGN oracle. If the challenge bit b in game $\mathrm{SIM}^{\mathcal{A}_1}_{\mathsf{WS},\mathsf{R}}(\lambda)$ is 1 then adversary $\mathcal{A}_1$ simulates game G_0 for adversary $\mathcal{A}$, and if $b = 0$ then adversary $\mathcal{A}_1$ simulates game G_1. We thus have Equation (1). To establish Equation (2), note that

$\underline{\mathcal{A}_1^{\textsc{Sign}}(1^\lambda, wp)}$

$S \leftarrow \emptyset\,;\ t \leftarrow 1\,;\ t^* \leftarrow \mathsf{KE.T}(\lambda) + 1$

$(pk, sk_1) \leftarrow_{\$} \mathsf{KE.Kg}(1^\lambda)\,;\ (pc, sc) \leftarrow sk_1$

$(i, m, \sigma) \leftarrow_{\$} \mathcal{A}^{\textsc{UpSim},\textsc{LkSim},\textsc{ExpSim},\textsc{SignSim}}(1^\lambda, (pk, wp), pc)$

$\mathsf{win}_1 \leftarrow (1 \le i < t^*) \wedge ((i, m, \sigma) \notin S)$

$\mathsf{win}_2 \leftarrow \mathsf{WS.Vf}(1^\lambda, wp, (1^\lambda, pk, i), m, \sigma)$

If $(\mathsf{win}_1 \wedge \mathsf{win}_2)$ then $b' \leftarrow 1$ else $b' \leftarrow 0$

Return b'

$\underline{\mathcal{A}_2^{\textsc{Sign}}(1^\lambda, wp)}$

$S \leftarrow \emptyset\,;\ t \leftarrow 1\,;\ t^* \leftarrow \mathsf{KE.T}(\lambda) + 1$

$(pk, sk_1) \leftarrow_{\$} \mathsf{KE.Kg}(1^\lambda)\,;\ (pc, sc) \leftarrow sk_1$

$(i, m, \sigma) \leftarrow_{\$} \mathcal{A}^{\textsc{UpSim},\textsc{LkSim},\textsc{ExpSim},\textsc{SignSim}}(1^\lambda, (pk, wp), pc)$

$x \leftarrow (1^\lambda, pk, i)$

Return (x, m, σ)

$\underline{\textsc{UpSim}()}$

If $t < \mathsf{KE.T}(\lambda)$ then

$\quad sk_{t+1} \leftarrow_{\$} \mathsf{KE.Up}(1^\lambda, pk, t, sk_t)$

$\quad (pc, sc) \leftarrow sk_{t+1}\,;\ t \leftarrow t + 1$

$\quad$ Return pc

Else return $\perp$

$\underline{\textsc{LkSim}(L)}$

$(pc, sc) \leftarrow sk_t\,;$ Return $L(sc)$

$\underline{\textsc{ExpSim}()}$

$t^* \leftarrow t\,;$ Return sk_t

$\underline{\textsc{SignSim}(m)}$

$\sigma \leftarrow_{\$} \textsc{Sign}((1^\lambda, pk, t), sk_t, m)$

$S \leftarrow S \cup \{(t, m, \sigma)\}\,;$ Return (t, σ)

Figure 9: Adversaries $\mathcal{A}_1$, $\mathcal{A}_2$ for proof of Theorem 5.1.

adversary $\mathcal{A}_2$ winning in game $\mathrm{EXT}_{\mathsf{WS,R}}^{\mathcal{A}_2}(\lambda)$ is equivalent to adversary $\mathcal{A}$ setting bad flag true in game G_1 by construction, except when $\mathcal{A}$ returns a forgery for time period $i \ge t^*$. In the latter case, $\mathcal{A}$ will not set bad flag true in G_1, whereas $\mathcal{A}_2$ may still win in its game.

Define PT adversary $\mathcal{B}$ against FOWCL-security of KE as follows:

$\underline{\mathcal{B}^{\textsc{Up},\textsc{Lk},\textsc{Exp}}(1^\lambda, pk, pc)}$

$t \leftarrow 1\,;\ (wp, std, xtd) \leftarrow_{\$} \mathsf{WS.SimPg}(1^\lambda)$

$(i, m, \sigma) \leftarrow_{\$} \mathcal{A}^{\textsc{UpSim},\textsc{LkSim},\textsc{ExpSim},\textsc{SignSim}}(1^\lambda, (pk, wp), pc)$

$sk \leftarrow_{\$} \mathsf{WS.Ext}(1^\lambda, wp, (1^\lambda, pk, i), xtd, m, \sigma)$

Return (i, sk)

$\underline{\textsc{SignSim}(m)}$

$\sigma \leftarrow_{\$} \mathsf{WS.SimSig}(1^\lambda, wp, (1^\lambda, pk, t), std, m)$

Return (t, σ)

$\underline{\textsc{UpSim}()}$

If $t < \mathsf{KE.T}(\lambda)$ then

$\quad t \leftarrow t + 1$

Return $\textsc{Up}()$

$\underline{\textsc{LkSim}(L)}$

Return $\textsc{Lk}(L)$

$\underline{\textsc{ExpSim}()}$

Return $\textsc{Exp}()$

Adversary $\mathcal{B}$ simulates $\mathcal{A}$'s oracles $\textsc{Up}, \textsc{Lk}, \textsc{Exp}$ via its own oracles of the same names. It answers $\mathcal{A}$'s $\textsc{Sign}$ queries using simulated signatures generated under the simulation trapdoor that it has itself picked. When $\mathcal{A}$ returns a forgery, $\mathcal{B}$ runs the witness extractor to learn the secret key (witness) and returns it as an output value. Adversary $\mathcal{B}$ is valid and δ-bounded because it makes the same $\textsc{Up}$, $\textsc{Lk}$ and $\textsc{Exp}$ queries as $\mathcal{A}$, whereas $\mathcal{A}$ is both valid and δ-bounded. (Note that the use of simulated parameters and signatures in games G_1 and G_2 results in a different environment than the one $\mathcal{A}$ may expect in game $\mathrm{FUFCL}_{\mathsf{KES}}^{\mathcal{A}}(\lambda)$. But, as per convention outlined in Section 3, adversary $\mathcal{A}$ is valid and δ-bounded regardless of its inputs and how its oracle queries are answered. Alternatively, one could ensure this by checking both conditions in code of games G_0–G_2.) To establish Equation (3), note that adversary $\mathcal{B}$ winning in game $\mathrm{FOWCL}_{\mathsf{KE}}^{\mathcal{B}}(\lambda)$ is equivalent to adversary $\mathcal{A}$ winning in game G_2 by construction, except when $\mathcal{A}$ returns an invalid forgery (containing either an invalid signature, or a message-signature pair learned as a result of calling its $\textsc{Sign}$ oracle). If the forgery is not valid, $\mathcal{A}$ will not win in G_2, but $\mathcal{B}$ may still win in $\mathrm{FOWCL}_{\mathsf{KE}}^{\mathcal{B}}(\lambda)$. ∎

Game $\mathrm{XS}^{\mathcal{A},\mathcal{E}}_{\mathsf{WE},\mathsf{R}}(\lambda)$
$b \leftarrow\$ \{0,1\}$
$(x, m_0, m_1, state) \leftarrow\$ \mathcal{A}_1(1^\lambda)$
$\mathsf{win}_1 \leftarrow (x \in \mathcal{L}(\mathsf{R})) \wedge (\lvert m_0\rvert = \lvert m_1\rvert)$
$c \leftarrow\$ \mathsf{WE.Enc}(1^\lambda, x, m_b)$
$b' \leftarrow\$ \mathcal{A}_2(1^\lambda, state, c)$
$\mathsf{win}_2 \leftarrow (b' = b)$
$w \leftarrow\$ \mathcal{E}(1^\lambda, x, m_0, m_1, state, c)$
$\mathsf{win}_3 \leftarrow (\text{not } \mathsf{R.Vf}(x, w))$
Return $(\mathsf{win}_1 \wedge \mathsf{win}_2 \wedge \mathsf{win}_3)$

Figure 10: Game defining extractable security of witness encryption scheme WE for **NP**-relation R.

5.2 FINDCL encryption from FOWCL key-evolution schemes

In this section we show that an FINDCL key-evolving encryption scheme can be constructed from any FOWCL key-evolution scheme, assuming the existence of a weak form of extractable witness encryption [26, 6, 28]. A witness encryption scheme for an **NP**-relation R allows to encrypt a message to an instance $x \in \{0,1\}^*$. To decrypt the resulting ciphertext, one has to provide a witness w such that (x, w) belongs to R. Extractable security of witness encryption requires that no PT adversary can distinguish between encryptions of any two equal-length messages m_0, m_1 to an instance x, unless a valid witness w can be efficiently recovered from x.

Given any FOWCL key-evolution scheme KE, we assume an extractable witness encryption scheme for a single **NP**-relation R = KE-REL[KE] as defined in Section 5.1. The relation R maps any triple $(1^\lambda, pk, i)$ containing a security parameter, a public key and a time period associated to KE to all corresponding valid secret keys sk_i as per the verification algorithm of this scheme. Here $(1^\lambda, pk, i)$ is an instance to which a message can be encrypted, and sk_i is a witness that would allow to decrypt the resulting ciphertext.

Garg, Gentry, Halevi and Wichs [25] showed that there is an **NP**-relation for which there exist no extractable witness encryption schemes, using a novel *special-purpose obfuscation* assumption introduced in their work. As a result, it is implausible that there is an extractable witness encryption scheme that is secure for *all* **NP**-relations. However, we require extractable security for a *single* **NP**-relation (different from the one used in [25]), thus evading their negative result. We note that this is nonetheless a very strong assumption, and our result in this section is interesting mostly in the context of building FS+CL primitives in a unified way.

We do not require the witness encryption scheme to provide any security in the case when a message is encrypted for an instance $(1^\lambda, pk, i)$ that is outside the relation, meaning when the former does not correspond to any valid secret key sk_i. This does not help to evade the implausibility result of [25], but it contrasts with the prior work on witness encryption that predominantly discusses the notion of soundness security that is only required to hold when encrypting messages for instances *outside* the relation [26, 6].

Witness encryption schemes. Let R be an **NP**-relation. A witness encryption scheme WE for R specifies PT algorithms WE.Enc and WE.Dec, where WE.Dec is deterministic. Encryption algorithm WE.Enc takes 1^λ, an instance $x \in \{0,1\}^*$ and a message $m \in \{0,1\}^*$ to return a ciphertext c. Decryption algorithm WE.Dec takes a witness w and ciphertext c to return $m \in \{0,1\}^* \cup \{\bot\}$.

$\underline{\mathsf{KEE.Kg}(1^\lambda)}$	$\underline{\mathsf{KEE.Enc}(1^\lambda, pk, i, m)}$
$(pk, sk_1) \leftarrow\!\!\$\ \mathsf{KE.Kg}(1^\lambda)$	$x \leftarrow (1^\lambda, pk, i)$
Return $(pk, , sk_1)$	$c \leftarrow\!\!\$\ \mathsf{WE.Enc}(1^\lambda, x, m)$
$\underline{\mathsf{KEE.Up}(1^\lambda, pk, i, sk_i)}$	Return (i, c)
Return $\mathsf{KE.Up}(1^\lambda, pk, i, sk_i)$	$\underline{\mathsf{KEE.Dec}(1^\lambda, pk, i, sk_i, (j, c))}$
	$w \leftarrow sk_i$
	Return $\mathsf{WE.Dec}(w, c)$

Figure 11: Definition of key-evolving encryption scheme $\mathsf{KEE} = \mathsf{WITNESS\text{-}KEE}[\mathsf{KE}, \mathsf{WE}, \ell]$.

Correctness requires that $\mathsf{WE.Dec}(w, c) = m$ for all $\lambda \in \mathbb{N}$, all $m \in \{0, 1\}^*$, all $x \in \mathcal{L}(\mathsf{R})$, all $w \in \mathsf{R}(x)$ and all $c \in [\mathsf{WE.Enc}(1^\lambda, x, m)]$.

<u>Extractable witness encryption.</u> Consider game XS of Fig. 10 associated to an **NP**-relation R, a witness encryption scheme WE for R, an adversary $\mathcal{A}$ and an extractor $\mathcal{E}$. For $\lambda \in \mathbb{N}$ let $\mathsf{Adv}^{\mathsf{xs}}_{\mathsf{WE},\mathsf{R},\mathcal{A},\mathcal{E}}(\lambda) = 2\Pr[\mathrm{XS}^{\mathcal{A},\mathcal{E}}_{\mathsf{WE},\mathsf{R}}(\lambda)] - 1$. We say that WE is XS[R]-secure if for every PT adversary $\mathcal{A}$ there exists a PT extractor $\mathcal{E}$ such that $\mathsf{Adv}^{\mathsf{xs}}_{\mathsf{WE},\mathsf{R},\mathcal{A},\mathcal{E}}(\cdot)$ is negligible. Our security definition is adapted from [6], weakening it to require that $\mathcal{A}$ chooses an instance x that belongs to $\mathcal{L}(\mathsf{R})$.

<u>Key-evolving encryption scheme WITNESS-KEE.</u> Let $\ell \colon \mathbb{N} \to \mathbb{N}$ be a polynomial. Let KE be a key-evolution scheme. Let $\mathsf{R} = \mathsf{KE\text{-}REL}[\mathsf{KE}]$ be the **NP**-relation as defined in Section 5.1. Let WE be a witness encryption scheme for R. We build a key-evolving encryption scheme $\mathsf{KEE} = \mathsf{WITNESS\text{-}KEE}[\mathsf{KE}, \mathsf{WE}, \ell]$ as defined in Fig. 11, where $\mathsf{KEE.skl} = \mathsf{KE.skl}$, $\mathsf{KEE.pcl} = \mathsf{KE.pcl}$, $\mathsf{KEE.scl} = \mathsf{KE.scl}$, $\mathsf{KEE.ml} = \ell$ and $\mathsf{KEE.T} = \mathsf{KE.T}$.

We show that if KE is FOWCL-secure and WE is XS[R]-secure for $\mathsf{R} = \mathsf{KE\text{-}REL}[\mathsf{KE}]$, then KEE is FINDCL-secure.

Theorem 5.2 *Let $\delta : \mathbb{N} \to [0, 1]$. Let $\ell \colon \mathbb{N} \to \mathbb{N}$ be a polynomial. Let KE be a δ-FOWCL key-evolution scheme. Let $\mathsf{R} = \mathsf{KE\text{-}REL}[\mathsf{KE}]$ be the **NP**-relation as defined in Section 5.1. Let WE be a witness encryption scheme for R, and assume that WE is XS[R]-secure. Then key-evolving encryption scheme $\mathsf{KEE} = \mathsf{WITNESS\text{-}KEE}[\mathsf{KE}, \mathsf{WE}, \ell]$ is δ-FINDCL secure.*

To prove this theorem, we assume for a contradiction that there exists a PT adversary that breaks the FINDCL-security of KEE. We use it to build a PT adversary $\mathcal{B}$ against the XS[R]-security of WE. We then consider two cases. If there does not exist a PT extractor $\mathcal{E}$ such that $\mathsf{Adv}^{\mathsf{xs}}_{\mathsf{WE},\mathsf{R},\mathcal{B},\mathcal{E}}(\cdot)$ is negligible, then WE is not XS[R]-secure, which contradicts our assumption about WE. Otherwise, there exists a PT extractor $\mathcal{E}$ that succeeds to recover the witness with a non-negligible probability, and we use it to break the FOWCL-security of KE.

Proof of Theorem 5.2: Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be a valid, δ-bounded PT adversary attacking KEE in game FINDCL. We build a PT adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ against the XS[R]-security of WE such that for every PT extractor $\mathcal{E}$ there exists a valid, δ-bounded PT adversary $\mathcal{I}$ against the FOWCL-security of KE giving

$$\mathsf{Adv}^{\mathsf{findcl}}_{\mathsf{KEE},\mathcal{A}}(\lambda) \le \mathsf{Adv}^{\mathsf{xs}}_{\mathsf{WE},\mathsf{R},\mathcal{B},\mathcal{E}}(\lambda) + 2 \cdot \mathsf{Adv}^{\mathsf{fowcl}}_{\mathsf{KE},\mathcal{I}}(\lambda)$$

for all $\lambda \in \mathbb{N}$.

We build a PT adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ against the XS[R]-security of WE as defined in Fig. 12. Algorithm $\mathcal{B}_1$ generates its own keys for the key evolution scheme KE and uses them to answer $\mathcal{A}_1$'s queries to oracles Up, Lk and Exp. When $\mathcal{A}_1$ chooses challenge messages m_0, m_1, algorithm $\mathcal{B}_1$

```
B_1(1^λ)
t ← 1
(pk, sk_1) ←$ KE.Kg(1^λ) ; (pc, sc) ← sk_1
(i, m_0, m_1, state) ←$ A_1^{UpSim,LkSim,ExpSim}(1^λ, pk, pc)
x ← (1^λ, pk, i)
Return (x, m_0, m_1, (state, i))

B_2(1^λ, (state, i), c)
b' ←$ A_2(1^λ, state, (i, c))
Return b'

UpSim()
If t < KE.T(λ) then
   sk_{t+1} ←$ KE.Up(1^λ, pk, t, sk_t)
   (pc, sc) ← sk_{t+1} ; t ← t + 1
   Return pc
Else return ⊥

LkSim(L)
(pc, sc) ← sk_t ; Return L(sc)

ExpSim()
Return sk_t
```

Figure 12: Adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ for proof of Theorem 5.2.

```
Games G_0–G_1
b ←$ {0,1} ; t ← 1 ; t* ← KE.T(λ) + 1
(pk, sk_1) ←$ KE.Kg(1^λ) ; (pc, sc) ← sk_1
(i, m_0, m_1, state) ←$ A_1^{Up,Lk,Exp}(1^λ, pk, pc)
If not (1 ≤ i < t*) then return false
If |m_0| ≠ |m_1| then return false
x ← (1^λ, pk, i) ; c ←$ WE.Enc(1^λ, x, m_b)
b' ←$ A_2(1^λ, state, (i, c))
w ←$ E(1^λ, x, m_0, m_1, (state, i), c)
If R.Vf(x, w) then
   bad ← true
   Return false                              // G_0
Return (b' = b)

Up()
If t < KE.T(λ) then
   sk_{t+1} ←$ KE.Up(1^λ, pk, t, sk_t)
   (pc, sc) ← sk_{t+1} ; t ← t + 1
   Return pc
Else return ⊥

Lk(L)
(pc, sc) ← sk_t ; Return L(sc)

Exp()
t* ← t ; Return sk_t
```

Figure 13: Games for proof of Theorem 5.2.

requests the corresponding challenge ciphertext in the witness extractability game against WE, R. Algorithm $\mathcal{B}_1$ passes the rest of $\mathcal{A}_1$'s output over to $\mathcal{B}_2$ which runs $\mathcal{A}_2$ on this data along with the received challenge ciphertext.

Let $\mathcal{E}$ be any PT extractor. Consider games G_0, G_1 of Fig. 13. Lines not annotated with comments are common to all games. Game G_1 is equivalent to game $\text{FINDCL}^{\mathcal{A}}_{\mathsf{KEE}}(\lambda)$, with the code of KEE expanded according to its definition, so we have $\Pr[G_1] = \Pr[\text{FINDCL}^{\mathcal{A}}_{\mathsf{KEE}}(\lambda)]$ for all $\lambda \in \mathbb{N}$. Game G_0 is equivalent to game $\text{XS}^{\mathcal{B},\mathcal{E}}_{\mathsf{WE},\mathsf{R}}(\lambda)$, except when $\mathcal{A}_1$ in G_0 returns a pair of challenge messages for time period $i \geq t^*$. In this case, $\mathcal{A}$ can not win in G_0, but $\mathcal{B}$ can still win in its game. We have

$\Pr[G_0] \leq \Pr[\mathrm{XS}^{\mathcal{B},\mathcal{E}}_{\mathsf{WE},\mathsf{R}}(\lambda)]$ for all $\lambda \in \mathbb{N}$.

We now build a valid, δ-bounded PT adversary $\mathcal{I}$ against the FOWCL-security of KE such that

$$\Pr[G_0(\lambda) \text{ sets } \mathsf{bad}] \leq \mathsf{Adv}^{\mathsf{fowcl}}_{\mathsf{KE},\mathcal{I}}(\lambda)$$

for all $\lambda \in \mathbb{N}$. The construction is as follows:

$$\begin{array}{l}
\underline{\mathcal{I}^{\textsc{Up},\textsc{Lk},\textsc{Exp}}(1^\lambda, pk, pc)} \\
b \leftarrow\$ \{0,1\} \\
(i, m_0, m_1, state) \leftarrow\$ \mathcal{A}_1^{\textsc{Up},\textsc{Lk},\textsc{Exp}}(1^\lambda, pk, pc) \\
x \leftarrow (1^\lambda, pk, i)\ ;\ c \leftarrow\$ \mathsf{WE.Enc}(1^\lambda, x, m_b) \\
w \leftarrow\$ \mathcal{E}(1^\lambda, x, m_0, m_1, (state, i), c) \\
\text{Return } (i, w)
\end{array}$$

Adversary $\mathcal{I}$ uses its inputs pk, pc in game $\mathrm{FOWCL}^{\mathcal{I}}_{\mathsf{KE}}(\lambda)$ to simulate $\mathcal{A}_1$. It samples its own challenge bit b, encrypts the corresponding challenge message received from $\mathcal{A}_1$, and runs the witness extractor algorithm $\mathcal{E}$ in an attempt to recover a witness for instance $(1^\lambda, pk, i)$. Any corresponding witness in relation R is a valid secret key sk_i for KE, meaning that adversary $\mathcal{I}$ breaks the FOWCL security of KE whenever $\mathcal{E}$ succeeds to recover a witness. The later event in game G_0 sets bad flag to true, yielding the above equation.

Adversary $\mathcal{I}$ is valid and δ-bounded because it makes the same $\textsc{Up}$, $\textsc{Lk}$ and $\textsc{Exp}$ queries as $\mathcal{A}_1$, whereas $\mathcal{A}$ is both valid and δ-bounded. (Note that game $\mathrm{FINDCL}^{\mathcal{A}}_{\mathsf{KEE}}$ is perfectly simulated for $\mathcal{A}_1$, so it is guaranteed to make at most $\delta(\lambda) \cdot \mathsf{KEE.scl}(\lambda)$ queries to its $\textsc{Lk}$ oracle per time period, regardless of the convention about adversary restrictions from Section 3.)

Games G_0 and G_1 are identical until bad, so by the Fundamental Lemma of Game-Playing [9] we have

$$\Pr[G_1(\lambda)] - \Pr[G_0(\lambda)] \leq \Pr[G_0(\lambda) \text{ sets } \mathsf{bad}]$$

for all $\lambda \in \mathbb{N}$. Combining all of the above, it follows that:

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{findcl}}_{\mathsf{KEE},\mathcal{A}}(\lambda) &= 2\Pr[\mathrm{FINDCL}^{\mathcal{A}}_{\mathsf{KEE}}(\lambda)] - 1 \\
&= 2\Pr[G_1(\lambda)] - 1 \\
&= 2\left((\Pr[G_1(\lambda)] - \Pr[G_0(\lambda)]) + \Pr[G_0(\lambda)]\right) - 1 \\
&\leq 2\left(\Pr[G_0(\lambda) \text{ sets } \mathsf{bad}] + \Pr[\mathrm{XS}^{\mathcal{B},\mathcal{E}}_{\mathsf{WE},\mathsf{R}}(\lambda)]\right) - 1 \\
&\leq 2\,\mathsf{Adv}^{\mathsf{fowcl}}_{\mathsf{KE},\mathcal{I}}(\lambda) + \mathsf{Adv}^{\mathsf{xs}}_{\mathsf{WE},\mathsf{R},\mathcal{B},\mathcal{E}}(\lambda)
\end{aligned}$$

for all $\lambda \in \mathbb{N}$. This establishes the theorem statement. ▌

References

[1] M. Abdalla and L. Reyzin. A new forward-secure digital signature scheme. In T. Okamoto, editor, *Advances in Cryptology – ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 116–129. Springer, Heidelberg, Dec. 2000. 6

[2] A. Akavia, S. Goldwasser, and V. Vaikuntanathan. Simultaneous hardcore bits and cryptography against memory attacks. In O. Reingold, editor, *TCC 2009: 6th Theory of Cryptography Conference*, volume 5444 of *Lecture Notes in Computer Science*, pages 474–495. Springer, Heidelberg, Mar. 2009. 3, 6

[3] J. Alwen, Y. Dodis, and D. Wichs. Leakage-resilient public-key cryptography in the bounded-retrieval model. In S. Halevi, editor, *Advances in Cryptology – CRYPTO 2009*, volume 5677 of *Lecture Notes in Computer Science*, pages 36–54. Springer, Heidelberg, Aug. 2009. 6

[4] P. Ananth, V. Goyal, and O. Pandey. Interactive proofs under continual memory leakage. In J. A. Garay and R. Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part II*, volume 8617 of *Lecture Notes in Computer Science*, pages 164–182. Springer, Heidelberg, Aug. 2014. 3, 6

[5] R. Anderson. Two remarks on public key cryptology, 1997. http://www. cl. cam. ac. uk/users/rja14. 2

[6] M. Bellare and V. T. Hoang. Adaptive witness encryption and asymmetric password-based cryptography. In J. Katz, editor, *PKC 2015: 18th International Conference on Theory and Practice of Public Key Cryptography*, volume 9020 of *Lecture Notes in Computer Science*, pages 308–331. Springer, Heidelberg, Mar. / Apr. 2015. 5, 14, 20, 21

[7] M. Bellare, S. Meiklejohn, and S. Thomson. Key-versatile signatures and applications: RKA, KDM and joint enc/sig. In P. Q. Nguyen and E. Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 496–513. Springer, Heidelberg, May 2014. 5, 14, 16, 17

[8] M. Bellare and S. K. Miner. A forward-secure digital signature scheme. In M. J. Wiener, editor, *Advances in Cryptology – CRYPTO'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 431–448. Springer, Heidelberg, Aug. 1999. 2, 4, 6, 9, 10

[9] M. Bellare and P. Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In S. Vaudenay, editor, *Advances in Cryptology – EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 409–426. Springer, Heidelberg, May / June 2006. 6, 17, 23

[10] X. Boyen, H. Shacham, E. Shen, and B. Waters. Forward-secure signatures with untrusted update. In A. Juels, R. N. Wright, and S. Vimercati, editors, *ACM CCS 06: 13th Conference on Computer and Communications Security*, pages 191–200. ACM Press, Oct. / Nov. 2006. 6

[11] E. Boyle, G. Segev, and D. Wichs. Fully leakage-resilient signatures. In K. G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 89–108. Springer, Heidelberg, May 2011. 3, 6, 9

[12] E. Boyle, G. Segev, and D. Wichs. Fully leakage-resilient signatures. *Journal of Cryptology*, 26(3):513–558, July 2013. 9

[13] Z. Brakerski and S. Goldwasser. Circular and leakage resilient public-key encryption under subgroup indistinguishability - (or: Quadratic residuosity strikes back). In T. Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 1–20. Springer, Heidelberg, Aug. 2010. 6

[14] Z. Brakerski, Y. T. Kalai, J. Katz, and V. Vaikuntanathan. Overcoming the hole in the bucket: Public-key cryptography resilient to continual memory leakage. In *51st Annual Symposium on Foundations of Computer Science*, pages 501–510. IEEE Computer Society Press, Oct. 2010. 3, 4, 6, 7, 9, 10

[15] R. Canetti, S. Halevi, and J. Katz. A forward-secure public-key encryption scheme. In E. Biham, editor, *Advances in Cryptology – EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 255–271. Springer, Heidelberg, May 2003. 2, 5, 10, 12

[16] M. Chase and A. Lysyanskaya. On signatures of knowledge. In C. Dwork, editor, *Advances in Cryptology – CRYPTO 2006*, volume 4117 of *Lecture Notes in Computer Science*, pages 78–96. Springer, Heidelberg, Aug. 2006. 5, 14, 16

[17] D. Dachman-Soled, S. D. Gordon, F.-H. Liu, A. O'Neill, and H.-S. Zhou. Leakage-resilient public-key encryption from obfuscation. In C.-M. Cheng, K.-M. Chung, G. Persiano, and B.-Y. Yang, editors, *PKC 2016: 19th International Conference on Theory and Practice of Public Key Cryptography, Part II*, volume 9615 of *Lecture Notes in Computer Science*, pages 101–128. Springer, Heidelberg, Mar. 2016. 3, 6

[18] Ö. Dagdelen and D. Venturi. A second look at Fischlin's transformation. In D. Pointcheval and D. Vergnaud, editors, *AFRICACRYPT 14: 7th International Conference on Cryptology in Africa*, volume 8469 of *Lecture Notes in Computer Science*, pages 356–376. Springer, Heidelberg, May 2014. 3, 6

[19] Y. Dodis, K. Haralambiev, A. López-Alt, and D. Wichs. Cryptography against continuous memory attacks. In *51st Annual Symposium on Foundations of Computer Science*, pages 511–520. IEEE Computer Society Press, Oct. 2010. 3, 4, 5, 6, 7, 9, 14

[20] Y. Dodis, K. Haralambiev, A. López-Alt, and D. Wichs. Efficient public-key cryptography in the presence of key leakage. In M. Abe, editor, *Advances in Cryptology – ASIACRYPT 2010*, volume 6477 of *Lecture Notes in Computer Science*, pages 613–631. Springer, Heidelberg, Dec. 2010. 6, 16

[21] Y. Dodis, Y. T. Kalai, and S. Lovett. On cryptography with auxiliary input. In M. Mitzenmacher, editor, *41st Annual ACM Symposium on Theory of Computing*, pages 621–630. ACM Press, May / June 2009. 3, 6

[22] Y. Dodis, A. B. Lewko, B. Waters, and D. Wichs. Storing secrets on continually leaky devices. In R. Ostrovsky, editor, *52nd Annual Symposium on Foundations of Computer Science*, pages 688–697. IEEE Computer Society Press, Oct. 2011. 6

[23] S. Faust, E. Kiltz, K. Pietrzak, and G. N. Rothblum. Leakage-resilient signatures. In D. Micciancio, editor, *TCC 2010: 7th Theory of Cryptography Conference*, volume 5978 of *Lecture Notes in Computer Science*, pages 343–360. Springer, Heidelberg, Feb. 2010. 10, 12, 29

[24] S. Faust, M. Kohlweiss, G. A. Marson, and D. Venturi. On the non-malleability of the Fiat-Shamir transform. In S. D. Galbraith and M. Nandi, editors, *Progress in Cryptology - INDOCRYPT 2012: 13th International Conference in Cryptology in India*, volume 7668 of *Lecture Notes in Computer Science*, pages 60–79. Springer, Heidelberg, Dec. 2012. 6

[25] S. Garg, C. Gentry, S. Halevi, and D. Wichs. On the implausibility of differing-inputs obfuscation and extractable witness encryption with auxiliary input. In J. A. Garay and R. Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 518–535. Springer, Heidelberg, Aug. 2014. 5, 6, 20

[26] S. Garg, C. Gentry, A. Sahai, and B. Waters. Witness encryption and its applications. In D. Boneh, T. Roughgarden, and J. Feigenbaum, editors, *45th Annual ACM Symposium on Theory of Computing*, pages 467–476. ACM Press, June 2013. 5, 14, 20

[27] S. Garg, A. Jain, and A. Sahai. Leakage-resilient zero knowledge. In P. Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 297–315. Springer, Heidelberg, Aug. 2011. 3, 6, 9

[28] S. Goldwasser, Y. T. Kalai, R. A. Popa, V. Vaikuntanathan, and N. Zeldovich. How to run turing machines on encrypted data. In R. Canetti and J. A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 536–553. Springer, Heidelberg, Aug. 2013. 5, 14, 20

[29] V. Goyal, A. Jain, and D. Khurana. Witness signatures and non-malleable multi-prover zero-knowledge proofs. Cryptology ePrint Archive, Report 2015/1095, 2015. `http://eprint.iacr.org/2015/1095`. 5, 16

[30] G. Itkis and L. Reyzin. Forward-secure signatures with optimal signing and verifying. In J. Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 332–354. Springer, Heidelberg, Aug. 2001. 6

[31] J. Katz and V. Vaikuntanathan. Signature schemes with bounded leakage resilience. In M. Matsui, editor, *Advances in Cryptology – ASIACRYPT 2009*, volume 5912 of *Lecture Notes in Computer Science*, pages 703–720. Springer, Heidelberg, Dec. 2009. 3, 6

[32] H. Krawczyk. Simple forward-secure signatures from any signature scheme. In S. Jajodia and P. Samarati, editors, *ACM CCS 00: 7th Conference on Computer and Communications Security*, pages 108–115. ACM Press, Nov. 2000. 6

[33] A. B. Lewko, M. Lewko, and B. Waters. How to leak on key updates. In L. Fortnow and S. P. Vadhan, editors, *43rd Annual ACM Symposium on Theory of Computing*, pages 725–734. ACM Press, June 2011. 3, 6

[34] A. B. Lewko, Y. Rouselakis, and B. Waters. Achieving leakage resilience through dual system encryption. In Y. Ishai, editor, *TCC 2011: 8th Theory of Cryptography Conference*, volume 6597 of *Lecture Notes in Computer Science*, pages 70–88. Springer, Heidelberg, Mar. 2011. 3, 5, 6

[35] V. Lyubashevsky, A. Palacio, and G. Segev. Public-key cryptographic primitives provably as secure as subset sum. In D. Micciancio, editor, *TCC 2010: 7th Theory of Cryptography Conference*, volume 5978 of *Lecture Notes in Computer Science*, pages 382–400. Springer, Heidelberg, Feb. 2010. 6

[36] T. Malkin, D. Micciancio, and S. K. Miner. Efficient generic forward-secure signatures with an unbounded number of time periods. In L. R. Knudsen, editor, *Advances in Cryptology – EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 400–417. Springer, Heidelberg, Apr. / May 2002. 6

[37] T. Malkin, I. Teranishi, Y. Vahlis, and M. Yung. Signatures resilient to continual leakage on memory and computation. In Y. Ishai, editor, *TCC 2011: 8th Theory of Cryptography Conference*, volume 6597 of *Lecture Notes in Computer Science*, pages 89–106. Springer, Heidelberg, Mar. 2011. 3, 6, 9

[38] M. Naor and G. Segev. Public-key cryptosystems resilient to key leakage. In S. Halevi, editor, *Advances in Cryptology – CRYPTO 2009*, volume 5677 of *Lecture Notes in Computer Science*, pages 18–35. Springer, Heidelberg, Aug. 2009. 3, 6

[39] J. B. Nielsen, D. Venturi, and A. Zottarel. Leakage-resilient signatures with graceful degradation. In H. Krawczyk, editor, *PKC 2014: 17th International Conference on Theory and Practice of Public Key Cryptography*, volume 8383 of *Lecture Notes in Computer Science*, pages 362–379. Springer, Heidelberg, Mar. 2014. 3, 6

A Construction of KES-TREE

Let SIG be a key-evolving signature scheme. We define a new key-evolving signature scheme KES-TREE in two steps. First, we will use SIG to specify a single node of a binary tree, defining a scheme KES-NODE that is parameterized by SIG. We will then build a signature scheme KES-TREE by composing multiple KES-NODE[SIG] nodes into a binary tree. We stress that KES-NODE is an *auxiliary* scheme whose only purpose is to simplify the definition and analysis of the key-evolving signature scheme KES-TREE. We now start by defining KES-NODE.

Key-evolving signature wrapper-node KES-NODE. Let SIG be a key-evolving signature scheme with SIG.ml = SIG.pkl + 1. Let Fig. 14 define the node scheme NODE = KES-NODE[SIG]. We use KES-NODE to provide an abstraction for a single node of a binary tree. It encapsulates a public key pk, the current time period i (at any moment, different nodes may have different individual time periods, depending on when they were created), the corresponding secret key sk_i, and a signature (j, σ) of its public key under parent node's secret key (where j indicates the time period of parent node's secret key when it was used to produce this signature). For the interface of the node, we define the following procedures. NODE.Gen generates a node with a fresh key-pair containing a public key and a secret key. NODE.GetKeys returns the pair of keys that are used for the node, and NODE.GetPk returns only its public key. NODE.EraseSk erases the secret key, and NODE.Up evolves the secret key. NODE.Sign takes a message m to return its signature under the current secret key. NODE.GenChildren generates and returns two new nodes with independent, fresh keys.

Algorithm $\mathsf{NODE.Gen}(1^\lambda)$	Algorithm $\mathsf{NODE.Sign}(1^\lambda, \mathsf{node}, m)$
$(pk, sk_1) \leftarrow\$ \mathsf{SIG.Kg}(1^\lambda)$ Return $(pk, 1, sk_1, (\bot, \bot))$	$(pk, i, sk_i) \leftarrow \mathsf{NODE.GetKeys}(\mathsf{node})$ $(i, \sigma_m) \leftarrow\$ \mathsf{SIG.Sig}(1^\lambda, pk, i, sk_i, m)$ Return (i, σ_m)
Algorithm $\mathsf{NODE.GetKeys}(\mathsf{node})$ $(pk, i, sk_i, (j, \sigma)) \leftarrow \mathsf{node}$ Return (pk, i, sk_i)	Algorithm $\mathsf{NODE.GenChildren}(1^\lambda, \mathsf{node})$ $\mathsf{node}_l \leftarrow\$ \mathsf{NODE.Gen}(1^\lambda)$ $\mathsf{node}_r \leftarrow\$ \mathsf{NODE.Gen}(1^\lambda)$ $(pk_l, i_l, lsk) \leftarrow \mathsf{NODE.GetKeys}(\mathsf{node}_l)$ $(pk_r, i_r, rsk) \leftarrow \mathsf{NODE.GetKeys}(\mathsf{node}_r)$ $(i, \sigma_l) \leftarrow\$ \mathsf{NODE.Sign}(1^\lambda, \mathsf{node}, 0 \,\|\, pk_l)$ $(i, \sigma_r) \leftarrow\$ \mathsf{NODE.Sign}(1^\lambda, \mathsf{node}, 1 \,\|\, pk_r)$ $\mathsf{node}_l \leftarrow (pk_l, i_l, lsk, (i, \sigma_l))$ $\mathsf{node}_r \leftarrow (pk_r, i_r, rsk, (i, \sigma_r))$ Return $(\mathsf{node}_l, \mathsf{node}_r)$
Algorithm $\mathsf{NODE.GetPk}(\mathsf{node})$ $(pk, i, sk_i, (j, \sigma)) \leftarrow \mathsf{node}$ Return pk	
Algorithm $\mathsf{NODE.EraseSk}(\mathsf{node})$ $(pk, i, sk_i, (j, \sigma)) \leftarrow \mathsf{node}$ Return $(pk, \bot, (\bot, \bot), (j, \sigma))$	
Algorithm $\mathsf{NODE.Up}(1^\lambda, \mathsf{node})$ $(pk, i, sk_i, (j, \sigma)) \leftarrow \mathsf{node}$ $sk_{i+1} \leftarrow\$ \mathsf{SIG.Up}(1^\lambda, pk, i, sk_i)$ Return $(pk, i+1, sk_{i+1}, (j, \sigma))$	

Figure 14: Definition of key-evolving signature wrapper-node $\mathsf{NODE} = \mathsf{KES\text{-}NODE}[\mathsf{SIG}]$.

It uses the current node's secret key to sign each of the newly generated public keys (we prepend '0' to the public key of the left child, and we prepend '1' to the public key of the right child). The resulting signatures are stored in the corresponding child nodes.

NOTATION CONVENTIONS. We now introduce additional notation that will be useful for the definition of signature scheme KES-TREE (see Section 2 for base notation). We denote vectors by boldface lowercase letters, for example $\mathbf{x}$. We denote the number of coordinates of a vector $\mathbf{x}$ by $|\mathbf{x}|$. We denote the i-th bit of a string $x \in \{0,1\}^*$ by $x[i]$ for any $1 \le i \le |x|$, and the i-th coordinate of a vector $\mathbf{x}$ by $\mathbf{x}[i]$ for any $1 \le i \le |\mathbf{x}|$. If $x \in \{0,1\}^*$ is a string then $x[i,j]$ denotes the substring $x[i] \dots x[j]$ for any $1 \le i \le j \le |x|$. If $i, \ell \in \mathbb{N}$ such that $i < 2^\ell$ then $\langle i \rangle_\ell \in \{0,1\}^\ell$ denotes the ℓ-bit binary representation of i. We associate each position in a binary tree with a string $w \in \{0,1\}^*$; the root node is associated to an empty string ε. For any node associated with a string w, we use $w \,\|\, 0$ to denote the position of its left child and we use $w \,\|\, 1$ to denote the position of its right child. Fig. 3 shows three binary trees with each node labeled as described above. We use $\mathsf{map}[w]$ for $w \in \{0,1\}^*$ to store information about the binary tree node at the position that is associated to w. This notation is used for readability; it helps to avoid specifying a more detailed data structure for storing information about the binary tree and its nodes.

KEY-EVOLVING SIGNATURE SCHEME KES-TREE. Let SIG be any key-evolving signature scheme with $\mathsf{SIG.ml} = \mathsf{SIG.pkl} + 1$. Let $h : \mathbb{N} \to \mathbb{N}$ be a polynomial such that $2^{h(\lambda)-1} \le \mathsf{SIG.T}(\lambda)$ for all $\lambda \in \mathbb{N}$. We build a key-evolving signature scheme $\mathsf{KES} = \mathsf{KES\text{-}TREE}[\mathsf{SIG}, h]$ as follows. Let $\mathsf{NODE} = \mathsf{KES\text{-}NODE}[\mathsf{KES}]$, $\mathsf{KES.pkl} = \mathsf{SIG.pkl}$, $\mathsf{KES.ml} = \mathsf{SIG.ml}$ and let the algorithms associated to KES be defined according to Fig. 15. For any $\lambda \in \mathbb{N}$ we have $\mathsf{KES.T}(\lambda) = 2^{h(\lambda)}$ and $\mathsf{KES.scl}(\lambda) = \mathsf{SIG.scl}(\lambda) \cdot (h(\lambda)+1)$. Furthermore, $\mathsf{KES.pcl}(\lambda), \mathsf{KES.sigl}(\lambda) = \mathcal{O}(h(\lambda) \cdot \mathsf{SIG.pkl}(\lambda) + h(\lambda) \cdot \mathsf{SIG.sigl}(\lambda))$.

The secret component of the secret key for KES scheme consists of all secret keys that *descend* from the path going from the binary tree's root node to the currently active leaf node, as well as the secret key for the leaf node. The public component of the secret key stores all public keys and signatures that are associated to this path, including those that lie *on* this path.

The construction of KES-TREE specifies a number of auxiliary procedures that perform low-

Algorithm $\mathsf{KES.Kg}(1^\lambda)$

$\mathsf{node} \leftarrow\!\!\$\ \mathsf{NODE.Gen}(1^\lambda)$
$pk \leftarrow \mathsf{NODE.GetPk}(\mathsf{node})$
$\mathsf{map} \leftarrow \perp$; $\mathsf{map}[\varepsilon] \leftarrow \mathsf{node}$
$\mathsf{map} \leftarrow \mathsf{GenBranch}(1^\lambda, \mathsf{map}, \varepsilon)$
$sk_1 \leftarrow \mathsf{GetSkFromMap}(1^\lambda, \mathsf{map})$
Return (pk, sk_1)

Algorithm $\mathsf{KES.Sig}(1^\lambda, pk, i, sk_i, m)$

If not $(1 \le i \le \mathsf{KES.T}(\lambda))$ then return $\perp$
$\mathsf{map} \leftarrow \mathsf{GetMapFromSk}(1^\lambda, sk_i)$
$w \leftarrow \langle i-1 \rangle_{h(\lambda)}$
$(j, \sigma_m) \leftarrow\!\!\$\ \mathsf{NODE.Sign}(1^\lambda, \mathsf{map}[w], m)$
$\sigma \leftarrow \mathsf{BuildSignature}(\mathsf{map}, (j, \sigma_m), w)$
Return (i, σ)

Algorithm $\mathsf{KES.Up}(1^\lambda, pk, i, sk_i)$

If not $(1 \le i < \mathsf{KES.T}(\lambda))$ then return $\perp$
$\mathsf{map} \leftarrow \mathsf{GetMapFromSk}(1^\lambda, sk_i)$
$w \leftarrow \langle i-1 \rangle_{h(\lambda)}$
$p \leftarrow (\max d \in [h(\lambda)] \ : \ w[d] = 0)$
$\mathsf{map} \leftarrow\!\!\$\ \mathsf{EvolveKeys}(1^\lambda, \mathsf{map}, w, p)$
$\mathsf{map} \leftarrow \mathsf{EraseNodes}(1^\lambda, \mathsf{map}, w, p)$
$v \leftarrow w[1, p-1] \,\|\, 1$
$\mathsf{map} \leftarrow\!\!\$\ \mathsf{GenBranch}(1^\lambda, \mathsf{map}, v)$
$sk_{i+1} \leftarrow \mathsf{GetSkFromMap}(1^\lambda, \mathsf{map})$
Return sk_{i+1}

Algorithm $\mathsf{KES.Vf}(1^\lambda, pk, m, (i, \sigma))$

If not $(1 \le i \le \mathsf{KES.T}(\lambda))$ then return $\perp$
$(\boldsymbol{\sigma}, (j, \sigma_m)) \leftarrow \sigma$; $w \leftarrow \langle i-1 \rangle_{h(\lambda)}$
$\mathsf{valid} \leftarrow \mathsf{true}$
For $d = 1, \ldots, h(\lambda)$ do
  $(pk^*, j^*, \sigma^*) \leftarrow \boldsymbol{\sigma}[d]$; $m^* \leftarrow w[d] \,\|\, pk^*$
  $\mathsf{valid} \leftarrow \mathsf{valid} \wedge \mathsf{SIG.Vf}(1^\lambda, pk, m^*, (j^*, \sigma^*))$
  $pk \leftarrow pk^*$
$\mathsf{valid} \leftarrow \mathsf{valid} \wedge \mathsf{SIG.Vf}(1^\lambda, pk, m, (j, \sigma_m))$
Return valid

Procedure $\mathsf{GenBranch}(1^\lambda, \mathsf{map}, w)$

While $|w| < h(\lambda)$ do
  $(l, r) \leftarrow\!\!\$\ \mathsf{NODE.GenChildren}(1^\lambda, \mathsf{map}[w])$
  $\mathsf{map}[w \,\|\, 0] \leftarrow l$; $\mathsf{map}[w \,\|\, 1] \leftarrow r$
  $\mathsf{map}[w] \leftarrow \mathsf{NODE.EraseSk}(\mathsf{map}[w])$
  $w \leftarrow w \,\|\, 0$
Return map

Procedure $\mathsf{BuildSignature}(\mathsf{map}, (j, \sigma_m), w)$

For $d = 1, \ldots, h(\lambda)$ do
  $v \leftarrow w[1, d]$
  $(pk, i, sk_i, (j, \sigma)) \leftarrow \mathsf{map}[v]$
  $\boldsymbol{\sigma}[d] \leftarrow (pk, j, \sigma)$
Return $(\boldsymbol{\sigma}, (j, \sigma_m))$

Procedure $\mathsf{EvolveKeys}(1^\lambda, \mathsf{map}, w, p)$

For $d = 1, \ldots, p-1$ do
  If $w[d] = 0$ then
    $v \leftarrow w[1, d-1] \,\|\, 1$
    $\mathsf{map}[v] \leftarrow\!\!\$\ \mathsf{NODE.Up}(1^\lambda, \mathsf{map}[v])$
Return map

Procedure $\mathsf{EraseNodes}(1^\lambda, \mathsf{map}, w, p)$

For $d = p, \ldots, h(\lambda)$
  $v \leftarrow w[1, p]$; $\mathsf{map}[v] \leftarrow \perp$
Return map

Procedure $\mathsf{GetSkFromMap}(1^\lambda, \mathsf{map})$

... // Omitted.
$sk_i \leftarrow (pc, sc)$
Return sk_i

Procedure $\mathsf{GetMapFromSk}(1^\lambda, sk_i)$

$(pc, sc) \leftarrow sk_i$
... // Omitted.
Return map

Figure 15: Definition of key-evolving signature scheme $\mathsf{KES} = \mathsf{KES\text{-}TREE}[\mathsf{SIG}, h]$.

level operations on the tree. Given a tree node, GenBranch repeatedly generates two child nodes and moves down to the left child; this procedure is used to generate the tree nodes that lie on (and descend from) the path from the current node to the leftmost leaf node that is reachable from the current node. Procedure BuildSignature takes a label associated to some leaf node, along with a SIG signature that was produced using this node, and collects all SIG public keys and signatures that lie on the path from the root to this leaf; all of these entities together form a KES signature. Procedure EvolveKeys is called during every key-update call: it evolves the secret keys of all nodes that are a part of the secret key for the KES scheme. Procedure EraseNodes deletes binary tree nodes that were used for the previous time period and are no longer needed. Procedures GetSkFromMap and GetMapFromSk allow to change the representation of KES secret keys, transforming them between the *map* representation (mapping each valid label of a node to all available information about the

```
Adversary B^{Up,Lk,Sign}(1^λ, pk_ch, pc_ch)
p ←$ [2^{h(λ)+1} − 1] ; num ← 0
S ← ∅ ; t ← 1 ; t* ← KES.T(λ) + 1
(pk, sk_1) ←$ KES.Kg(1^λ) ; (pc, sc) ← sk_1
(i, m, σ) ←$ A^{UpSim,LkSim,ExpSim,SignSim}(1^λ, pk, pc)
(σ, (j, σ_m)) ← σ ; w ← ⟨i − 1⟩_{h(λ)}
For d = 1, ..., h(λ) do
   (pk*, j*, σ*) ← σ[d] ; m* ← w[d] || pk*
   If pk = pk_ch then return (j*, m*, σ*)
   pk ← pk*
Return (j, m, σ_m)

UpSim()
If t < KES.T(λ) then
   sk_{t+1} ←$ KES.Up(1^λ, pk, t, sk_t)
   (pc, sc) ← sk_{t+1} ; t ← t + 1
   Return pc
Else return ⊥

LkSim(L)
(pc, sc) ← sk_t ; L* ← L(sk_t, ·) ; Return Lk(L*)

ExpSim()
t* ← t ; Return sk_t

SignSim(m)
(t, σ) ←$ KES.Sig(1^λ, pk, t, sk_t, m)
S ← S ∪ {(t, m, σ)} ; Return (t, σ)
```

Figure 16: Adversary $\mathcal{B}$ for proof of Theorem 4.1.

corresponding node) and the *component* representation (organizing the information about all nodes into a public component and a secret component); we omit the (trivial but detailed) code of these procedures.

B Proof of Theorem 4.1

Let $\mathcal{A}$ be a valid, γ-bounded PT adversary attacking the FUFCL-security of KES. We build a valid, δ-bounded PT adversary $\mathcal{B}$ attacking the UFCL-security of SIG such that

$$\mathsf{Adv}^{\mathsf{ufcl}}_{\mathsf{SIG},\mathcal{B}}(\lambda) \geq \frac{\mathsf{Adv}^{\mathsf{fufcl}}_{\mathsf{KES},\mathcal{A}}(\lambda)}{2^{h(\lambda)+1}-1} \geq \frac{\mathsf{Adv}^{\mathsf{fufcl}}_{\mathsf{KES},\mathcal{A}}(\lambda)}{4\cdot \mathsf{SIG.T}(\lambda)-1} \tag{4}$$

for all $\lambda \in \mathbb{N}$. Our proof is similar to that of [23].

Informally, adversary $\mathcal{B}$ will be constructed as follows. It will simulate game FUFCL for $\mathcal{A}$, embedding its own challenge public-key pk_{ch} in a random position of the binary tree construction of KES = KES-TREE[SIG, h]. Specifically, out of all the $2^{h(\lambda)+1}-1$ nodes that are required to build KES (across all time periods), adversary $\mathcal{B}$ will itself generate the keys for $2^{h(\lambda)+1}-2$ of these nodes, and use them to answer $\mathcal{A}$'s oracle queries. The remaining single node (in a randomly chosen position) will correspond to a challenge key-pair $(pk_{\mathsf{ch}}, sk_{\mathsf{ch}})$ when attacking the UFCL-security of SIG. Although $\mathcal{B}$ will not know the secret key sk_{ch}, it will use its oracles Up, Lk, Sign to answer $\mathcal{A}$'s queries that require the knowledge of this secret key.

In order to win in game FUFCL, adversary $\mathcal{A}$ has to forge a valid KES signature for some time period i. This is only possible if $\mathcal{A}$ succeeds to forge a signature for a SIG public key that is associated to some node that belongs to the path from the root of the KES binary-tree construction to the leaf that is associated with time period i. Adversary $\mathcal{B}$ then returns this SIG signature as its own forgery. Adversary $\mathcal{B}$ breaks the UFCL-security of SIG if adversary $\mathcal{A}$ was succesful and if $\mathcal{B}$ managed to guess the exact node for which the forgery was going to happen, where the probability of the latter is $1/(2^{h(\lambda)+1}-1)$.

Note that $\mathcal{A}$ is γ-bounded for $\gamma(\cdot) = \delta(\cdot)/(h(\cdot)+1)$ and the secret component of the secret key in scheme KES is at most $h(\cdot)+1$ times larger than that of scheme SIG (its size varies depending on the time period), so the described construction of $\mathcal{B}$ will be δ-bounded.

We build adversary $\mathcal{B}$ attacking the UFCL-security of SIG as defined in Fig. 16. Adversary $\mathcal{B}$

Algorithm $\mathsf{NODE.Gen}(1^\lambda)$	Algorithm $\mathsf{NODE.Up}(1^\lambda, \mathsf{node})$	Algorithm $\mathsf{NODE.Sign}(1^\lambda, \mathsf{node}, m)$
$\mathsf{num} \leftarrow \mathsf{num} + 1$	$(pk, i, sk_i, (j, \sigma)) \leftarrow \mathsf{node}$	$(pk, i, sk_i) \leftarrow \mathsf{NODE.GetKeys}(\mathsf{node})$
If $\mathsf{num} = \mathsf{num}_{\mathsf{ch}}$ then	If $sk_i = \perp$ then	If $sk_i = \perp$ then
$\quad pk \leftarrow pk_{\mathsf{ch}}$; $sk_1 \leftarrow (pc_{\mathsf{ch}}, \perp)$	$\quad pc \leftarrow \textsc{Up}()$; $sk_{i+1} \leftarrow (pc, \perp)$	$\quad z \leftarrow\$ \textsc{Sign}(m)$
Else	Else	Else
$\quad (pk, sk_1) \leftarrow\$ \mathsf{SIG.Kg}(1^\lambda)$	$\quad sk_{i+1} \leftarrow\$ \mathsf{SIG.Up}(1^\lambda, pk, i, sk_i)$	$\quad z \leftarrow\$ \mathsf{SIG.Sig}(1^\lambda, pk, i, sk_i, m)$
Return $(pk, 1, sk_1, (\perp, \perp))$	Return $(pk, i+1, sk_{i+1}, (j, \sigma))$	$(i, \sigma_m) \leftarrow z$; Return (i, σ_m)

Figure 17: Overrided definitions of NODE.Gen, NODE.Up and NODE.Sign as implemented by adversary $\mathcal{B}$ for the simulation of adversary $\mathcal{A}$ in proof of Theorem 4.1.

---

simulates game $\mathrm{FUFCL}^{\mathcal{A}}_{\mathsf{KES}}(\lambda)$ for adversary $\mathcal{A}$, and for that purpose it has to implement the code of $\mathsf{KES} = \mathsf{KES\text{-}TREE}[\mathsf{SIG}, h]$. In order to avoid repeating the detailed definition of KES-TREE in its entirety, we only redefine the algorithms that are changed (compared to their original specifications). Specifically, we do not redefine any of the algorithms that are associated directly to $\mathsf{KES} = \mathsf{KES\text{-}TREE}[\mathsf{SIG}, h]$ (meaning KES.* and the associated procedures from Fig. 15), because all of them remain the same. For the auxiliary scheme NODE = KES-NODE[SIG] that is used by KES, we redefine algorithms that make use of the stored secret key. These algorithms are provided in Fig. 17.

Adversary $\mathcal{B}$ samples $p \leftarrow\$ [2^{h(\lambda)+1}-1]$ to denote a random position in the tree. It then maintains a variable num to count the number of nodes that were created this far, incrementing it prior to generating every next key pair for scheme SIG; this is done inside the redefined algorithm NODE.Gen in Fig. 17. If $\mathsf{num} = p$ then instead of sampling a new key pair, adversary sets pk_{ch} as the public key of the new node, it sets pc_{ch} as the public component of this node's secret key, and it uses $\perp$ to mark that the secret component of the secret key is unknown. Every time a secret key of some node has to be accessed, we consider two cases: Either the secret key is known, and hence it can be used directly. Or it is unknown (meaning it corresponds to the challenge public key pk_{ch}), in which case $\mathcal{B}$'s oracles are used to perform the necessary operations instead. This is done inside the redefined algorithms NODE.Up and NODE.Sign in Fig. 17.

In procedure LKSIM we use $L^* \leftarrow L(sk_t, \cdot)$ to denote the process of building a circuit L^* by hardwiring the input gates of circuit L to contain the known values of sk_t. As a result, circuit L^* may take only the challenge secret key sk_{ch} as input. Note that if sk_t does not contain the challenge node at the current time period, then L^* is a constant circuit with a single-bit output. We assume that a call to $\textsc{Lk}(L^*)$ returns a valid output in both cases.

According to the definition of KES, its signature consists of a chain of SIG signatures along the path in the binary tree of SIG nodes, going from its root node down to the leaf node that corresponds to the returned KES time period i. If the KES signature is valid, then for every two nearby nodes on this path, the public key of the parent's node is used to sign its child node's public key. If adversary $\mathcal{A}$ returns a KES forgery and any of the parent nodes have a public key that matches pk_{ch} then adversary returns the corresponding child's public key and its signature as a potential SIG forgery. Otherwise, it returns the last message-signature pair from this path. ▮