DSKE: Digital Signatures with Key Extraction

Zhipeng Wang[1], Orestis Alpos[2], Alireza Kavousi[3], Harry W. H. Wong[4], Sze Yiu Chau[4], Duc V. Le[5], and Christian Cachin[2]

[1] *Imperial College London*
[2] *University of Bern*
[3] *University College London*
[4] *The Chinese University of Hong Kong*
[5] *Visa Research*

Abstract

This work introduces DSKE, digital signatures with key extraction. In a DSKE scheme, the private key can be extracted if more than a threshold number of signatures on different messages are ever created while, within the threshold, each signature continues to authenticate the signed message. We propose a formal definition of DSKE, as well as two provably secure constructions, one from hash-based digital signatures and one from polynomial commitments.

We demonstrate that DSKE is useful for various applications, such as cryptographic deniability and spam prevention. First, we introduce the GroupForge signature scheme, leveraging DSKE to achieve deniability in digital communication. GroupForge integrates DSKE with a Merkle tree and timestamps to produce a forward-forgeable signature equipped with extractable sets, ensuring deniability under a fixed public key. We illustrate that GroupForge can serve as a viable alternative to Keyforge in the non-attributable email protocol of Specter, Park, and Green (USENIX Sec '21), thereby eliminating the need for continuous disclosure of outdated private keys. GroupForge can also operate as a short-lived signature, providing deniability non-interactively and agnostic to time. Second, we leverage the inherent extraction property of DSKE to develop a Rate-Limiting Nullifier (RLN) scheme. RLN efficiently identifies and expels spammers once they exceed a predetermined action threshold, thereby jeopardizing their private keys.

Moreover, we implement both variants of the DSKE to demonstrate their performance and show they are comparable to existing signature schemes. We also implement GroupForge from the polynomial commitment-based DSKE and illustrate the practicality of our proposed scheme.

1 Introduction

Digital signature schemes [GMR88] play an important role in protecting the integrity of data transmitted over the Internet. In some jurisdictions [Bly05, Mas16, Kar19], a digital signature applied to data can serve as evidence of the sender's authorship of the data. Moreover, the signature of a message remains valid until either the underlying signature scheme is broken or the private key is compromised. However, as pointed outby Borisov, Goldberg, and Brewer in their work on off-the-record (OTR) communication [BGB04], this "long-lived" property is unsuitable for certain types of messages [BCG$^+$23]. For instance, if Alice wishes to communicate privately with Bob, she can encrypt her messages using Bob's public key (i.e., for confidentiality) and sign them with her private key (i.e., for authenticity). However, if Eve compromises Bob's computer at some point in the future, Eve will be able to read all of Bob's previous messages from Alice, and use the signatures to prove to Judy that the messages indeed originated from Alice.

To address this problem, OTR messaging requires an interactive key agreement protocol between the sender and the recipient to agree on session keys before exchanging messages. Hence, participants can achieve plausible deniability by revealing the message authentication code (MAC) key at the end of the conversation; thus, no one can prove that any person is the author of the messages. However, this pair-wise key agreement required in OTR is not scalable for applications such as email protocols, in which there is often no prior end-to-end interaction among the participating parties.

Another way to achieve deniability is to simply require the sender to periodically rotate keys and publish their old private keys [Gre20]. This method enables anyone to forge signatures using the published private keys and thus offers deniability to old transcripts. In fact, this method is being suggested to offer deniability in domain keys identified mail (DKIM) [ACD$^+$07], where SMTP servers sign outgoing emails on behalf of the whole domain using a single key, as a way to safeguard against email spoofing. A server sending an email cryptographically signs it so that the recipient can verify that it has originated from the reported server. A side effect of this action is *email attributability* which stems from the fact that the digital signature remains valid for a long time, potentially forever [Gre20]. As a result, a malicious actor, who at any time gains access to these emails, can provably link them to their sender, which in turn incentivizes extortion and retaliation, among others.

Problem Statement. In this paper, we focus on answering the following question that arises naturally from the limitations of existing attempts [Gre20, BGB04, ABC22]:

Is it possible to design a signature scheme that allows the recipients to *verify the validity of the signature, while enabling the sender to gain plausible deniability, without requiring the constant publication of old key materials or any additional components?*

It is worth noting that the question itself is seemingly a contradiction due to the non-repudiation property of the digital signature. In this work, we propose digital signatures with key extraction (DSKE) scheme to answer the aforementioned question affirmatively. We first propose a general framework, showing that any one-time hash-based signature scheme, such as Lamport [Lam79] and Winternitz OTS [Mer89, BDE^{+}11], can be turned into a DSKE scheme, and present two concrete constructions. We then propose another DSKE construction based on polynomial commitments [KZG10]. This type of signature scheme comes with an *extractable set*, a set of signatures from which the private key can be extracted *asynchronously*.

Independently, the key extraction property of DKSE is particularly useful when there is a need to disincentivize or penalize the creation of more than a certain number of signatures. As concrete examples, one may consider the issue of double-spending [RKS] or double-signing on executable code [DRS18]. Both could be mitigated by putting the user under the threat of key leakage.[1]

Contributions. Our contributions can be summarized as follows:

- We formally define the notion of DSKE. It comes with an *extractable set*, a set of signatures that can be used to extract the private key.
- We introduce a framework to make one-time hash-based DSKE, and present two constructions for $\mathtt{DSKE}_{\mathsf{hash}}$. One is based on Lamport [Lam79] and the other is based on Winternitz signatures [Mer89, BDE^{+}11]. We demonstrate that generating multiple signatures while reusing a private key in one-time hash-based signatures results in an extractable set with an overwhelming probability.
- We propose a polynomial commitment-based DSKE, and present a concrete construction $\mathtt{DSKE}_{\mathsf{poly}}$ based on the KZG commitment scheme [KZG10]. In comparison to hash-based DSKE, $\mathtt{DSKE}_{\mathsf{poly}}$ offers shorter signature size and more flexibility to the signer in customizing the size of the extractable set.
- We use $\mathtt{DSKE}_{\mathsf{poly}}$ to present two concrete applications. First, we introduce GroupForge, a forward-forgeable signature with extractable set for deniability under a fixed public key. This makes GroupForge a potential replacement for Keyforge (USENIX Sec '21) [SPG21] with additional benefit like eliminating the reliance on a trusted party and the need for

[1]We remark that DSKE in this sense resembles an existing primitive in the literature, named double-authentication-preventing signatures (DAPS). However, there are crucial differences between the two primitives, as we elaborate in Section 9.

continual disclosure of outdated private keys. We show how GroupForge can also serve as a *short-lived signature* [ABC22] without using expensive timed cryptography. Second, we harness the inherent extraction property of DSKE to build a Rate-Limiting Nullifier (RLN) protocol, which can identify and expel spammers once they exceed a certain threshold in some action.

- We implement and evaluate both DSKE and GroupForge constructions, thereby highlighting their practicality.

2 Preliminaries

Notation. We denote by 1^λ the security parameter and by $\mathsf{negl}(\lambda)$ a negligible function of λ. We express by (pk, sk) a pair of public and private keys. We let $[n]$ denote the set $\{1, \ldots, n\}$. Moreover, we require that pk can always be efficiently derived from sk, and we denote $\mathsf{extractPK}(sk) = pk$ to be the deterministic function for doing so. For a field $\mathbb{F}$, we denote $\mathbb{F}^{\leq d}(X)$ the set of polynomials in $\mathbb{F}[X]$ with degree at most d. We denote by $\mathcal{M}$ the message space and $\mathcal{S}$ the signature space. For a non-negative integer j, we let $f^{(j)}$ be the j-th iterate of the function f, i.e., $f^{(j)}(x) = f(f(\cdots f(x)\cdots))$ where f is repetitively calculated j times. We use the notation $p(x) \leftarrow \mathsf{Interpolate}(\{x_i, y_i\}_{i\in[d+1]})$ to denote using Lagrange interpolation to obtain a degree-d polynomial given $d+1$ evaluation points and their corresponding evaluations.

Hash Functions. Our constructions employ the following standard properties of cryptographic hash functions. We use $H : \mathcal{K} \times \mathcal{M} \to \{0,1\}^\lambda$ to denote a family of hash functions that is parameterized by a key $k \in \mathcal{K}$ and message $m \in \mathcal{M}$ and outputs a binary string of length λ. For this work, we consider cryptographic hash functions [RS04], satisfying preimage resistance and collision resistance properties. In practice, the key for standard hash functions is public; therefore, from this point, we refer to the cryptographic hash function h sampled from a family of hash functions as a fixed function $H : \mathcal{M} \to \{0,1\}^\lambda$.

Polynomial Commitment Schemes. A polynomial commitment scheme (PCS) allows a prover to commit to a polynomial $f(X) \in \mathbb{F}^{\leq d}(X)$ and later open $f(X)$ at arbitrary points x, revealing only the value $f(x)$.

Definition 1 (Polynomial Commitment)**.** *A PCS consists of the following algorithms.*

- $(ck, vk) \leftarrow \mathsf{Setup}(1^\lambda, d)$*: The setup algorithm takes as input a security parameter λ, a maximum degree $d \in \mathbb{N}$, and outputs the public commitment key ck, which allows committing to polynomials in $\mathbb{F}^{\leq d}(X)$, and the public verification key vk.*

- $C_f \leftarrow \mathsf{Com}(ck, f(X))$*: The commitment algorithm takes as input the commitment key ck, a polynomial $f(X) \in \mathbb{F}^{\leq d}(X)$, and outputs a commitment $C_f \in G$ to the polynomial $f(X)$.*

- $(\pi, y) \leftarrow \mathsf{Open}(ck, C_f, x, f(X))$*: The algorithm takes as input a commitment key ck, a commitment C_f, an evaluation point x, the polynomial $f(X)$, and outputs $y = f(x) \in \mathbb{F}$ and a proof $\pi \in G$.*

- $0/1 \leftarrow \mathsf{Check}(vk, C_f, x, y, \pi)$*: The algorithm takes as input the verification key vk, the commitment C_f, a point x, the claimed evaluation y, the opening proof π, and outputs 1 iff $y = f(x)$.*

A PSC is secure if it can guarantee *correctness*, *computational hiding*, *evaluation binding*, *polynomial binding*, and *interpolation binding* properties from the polynomial commitment scheme.

Definition 2 (Correctness [KZG10]). *Let $(ck, vk) \leftarrow \mathsf{Setup}(1^\lambda, d)$, $f(X) \in \mathbb{F}^{\leq d}(X)$, and $C_f \leftarrow \mathsf{Com}(ck, f(X))$. Then for any (π, y) output by $\mathsf{Open}(ck, C_f, x, f(X))$, we have $\mathsf{Check}(vk, C_f, x, y, \pi) = 1$.*

Definition 3 (Computational Hiding [KZG10]). *Given $(ck, vk) \leftarrow \mathsf{Setup}(1^\lambda, d)$, the commitment $C_f = \mathsf{Com}(ck, f(X))$, and $\hat{d}$ valid openings (y_i, π_i) for points x_i, where $i \in \{1, \dots, \hat{d}\}$, $\hat{d}$ is the degree of $f(X)$, and $\hat{d} \leq d$, no PPT adversary can determine the value $f(x')$, for $x' \notin \{x_1, \dots, x_{\hat{d}}\}$, except with a negligible probability.*

Definition 4 (Evaluation Binding [KZG10]). *Given $(ck, vk) \leftarrow \mathsf{Setup}(1^\lambda, d)$, no PPT adversary can compute commitment C_f, point x, and two openings (π_1, y_1), (π_2, y_2) for x, such that $\mathsf{Check}(vk, C_f, x, y_1, \pi_1) = 1$, $\mathsf{Check}(vk, C_f, x, y_2, \pi_2) = 1$, and $y_1 \neq y_2$.*

Definition 5 (Polynomial Binding [KZG10]). *Given $(ck, vk) \leftarrow \mathsf{Setup}(1^\lambda, d)$, no PPT adversary can compute polynomials $f(X)$ and $f'(X)$, such that $f(X) \neq f'(X)$ and $\mathsf{Com}(ck, f(X)) = \mathsf{Com}(ck, f'(X))$.*

Abraham *et al.* [AJM$^+$23] recently introduced a new property termed *interpolation binding* for polynomial commitment schemes. This property implies that if a sufficient number of valid evaluations of the committed polynomial are provided, the polynomial interpolated from these points must be identical to the originally committed polynomial.

Definition 6 (Interpolation Binding [AJM$^+$23]). *Given $(ck, vk) \leftarrow \mathsf{Setup}(1^\lambda, d)$, no PPT adversary can output $(C_f, \{x_i, y_i, \pi_i\}_{i \in [d+1]})$ such that $x_j \neq x_k$ for all $j \neq k$, $\mathsf{Check}(vk, C_f, x_i, y_i, \pi_i) = 1$ for all $i \in [d+1]$, and $C_f \neq \mathsf{Com}(ck, f'(X))$ where $f'(X) = \mathsf{Interpolate}(\{x_i, y_i\}_{i \in [d+1]})$.*

3 Digital Signatures with Key Extraction (DSKE)

In this section, we formally define the notion of digital signatures with key extraction (DSKE). We adopt the standard digital signature definition and introduce a new algorithm to capture the capability of extracting the private key from a set of signatures.

Definition 7 ((k, δ)-Digital Signature with Key Extraction)**.** *A signature scheme Σ, with key extraction consists of five algorithms:*

- $par \leftarrow \mathsf{Setup}(1^\lambda)$*: The setup algorithm takes a security parameter* 1^λ *and outputs a set of public parameters* par*. This algorithm runs once, and the public parameters are implicitly input to all subsequent algorithms.*
- $(pk, sk) \leftarrow \mathsf{KeyGen}(par)$*: The key generation is a* probabilistic *algorithm that outputs a pair* (pk, sk) *of public and private keys.*
- $\sigma \leftarrow \mathsf{Sign}(sk, m)$*: The signing algorithm is a* probabilistic *algorithm that takes a private key* sk *and a message* $m \in \mathcal{M}$ *as input and outputs a signature* σ *in the signature space* $\mathcal{S}$*.*
- $b \leftarrow \mathsf{Verify}(pk, m, \sigma)$*: The verification algorithm is a* deterministic *algorithm that takes a public key* pk*, a message* m*, a signature* σ *as input, and outputs the validity of the signature,* $b \in \{0, 1\}$*.*
- $sk \leftarrow \mathsf{Extract}(\{(m_i, \sigma_i)\}_{i\in[k]}, pk)$*: The extraction algorithm is a* deterministic *algorithm that takes as input a set of distinct message-signature pairs* $(m_i, \sigma_i)_{i\in[k]}$ *and the public key* pk *such that* $\mathsf{Verify}(pk, m_i, \sigma_i) = 1$*, and outputs the underlying private key* sk *with probability* δ *and* $\perp$ *with probability* $1 - \delta$*.*

Besides the straightforward correctness definition, we consider other properties of DSKE: existential unforgeability (Definition 8), the extractability with trusted generation (Definition 9) and untrusted generation (Definition 10).

To define existential unforgeability, we consider the following experiment.

d-times signature experiment $\mathsf{SignExp}^d_{\mathcal{A},\Sigma}(\lambda)$.

1. $\mathsf{Setup}(1^\lambda)$ and $\mathsf{KeyGen}(par)$ are run to obtain keys (pk, sk).
2. $\mathcal{A}$ is given pk and can ask up to d queries to the signing oracle $\mathsf{Sign}(sk, \cdot)$. Let $Q_{\mathcal{A}}^{\mathsf{Sign}(sk,\cdot)} = \{m_i\}_{i\in[d]}$ be the set of all messages for which $\mathcal{A}$ queries $\mathsf{Sign}(sk, \cdot)$, where the i^{th} query is a message $m_i \in \mathcal{M}$. Eventually, $\mathcal{A}$ outputs a pair $(m^*, \sigma^*) \in \mathcal{M} \times \mathcal{S}$.
3. The output of the experiment is defined to be 1 if and only if $m^* \notin Q_{\mathcal{A}}^{\mathsf{Sign}(sk,\cdot)}$ and $\mathsf{Verify}(pk, m^*, \sigma^*) = 1$.

Definition 8 (Existential Unforgeability)**.** *A digital signature scheme Σ is* existentially unforgeable under a d-times adaptive chosen-message attack*, or* d-times-secure*, if for all PPT adversaries $\mathcal{A}$ the success probability in the d-times signature experiment is negligible:* $\Pr[\mathsf{SignExp}^{d}_{\mathcal{A},\Sigma}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$.

We proceed to define the extractability with trusted generation, in which the private key becomes extractable upon accumulating an adequate quantity of signatures generated by an honest signer. This property can be useful for applications where the signer wants to achieve deniability.

Definition 9 (Extractability with Trusted Generation)**.** *A digital signature scheme has a (k, δ)-extractable set when the algorithm* $\mathsf{Extract}(\cdot)$ *on input k distinct message-signature pairs $\{(m_i, \sigma_i)\}_{i\in[k]}$ and the public key pk, such that each σ_i is a valid signature on m_i under pk, outputs the private key sk with probability δ. That is:*

$$\Pr\left[\begin{array}{l} par \leftarrow \mathsf{Setup}(1^\lambda), (pk, sk) \leftarrow \mathsf{KeyGen}(par), \\ m_i \leftarrow \mathcal{M},\ \ s.t.\ m_i \neq m_j,\ \ for\ i, j \in [k], i \neq j, \\ \sigma_i \leftarrow \mathsf{Sign}(sk, m_i), \\ sk' \leftarrow \mathsf{Extract}\left(\{(m_i, \sigma_i)\}_{i\in[k]}, pk\right) \end{array} : pk = \mathsf{extractPK}(sk')\right] = \delta$$

We further define the extractability with untrusted generation, in which the signatures are generated by an adversary. Such property can be used to construct spam prevention by revealing the secret key. Intuitively, this property prevents an adversary from creating more message-signature pairs than the pre-defined threshold.

Definition 10 (Extractability with Untrusted Generation)**.** *A digital signature scheme has a (k, δ)-extractable set when the algorithm* $\mathsf{Extract}(\cdot)$ *on input k distinct message-signature pairs $\{(m_i, \sigma_i)\}_{i\in[k]}$ and the public key pk generated by an adversary $\mathcal{A}$, outputs the private key sk with probability δ:*

$$\Pr\left[\begin{array}{l} par \leftarrow \mathsf{Setup}(1^\lambda), (pk, \{(m_i, \sigma_i)\}_{i\in[k]}) \leftarrow \mathcal{A}(1^\lambda, par), \\ s.t.\ \mathsf{Verify}(pk, m_i, \sigma_i) = 1, \\ m_i \neq m_j,\ \ for\ i, j \in [k], i \neq j, \\ sk' \leftarrow \mathsf{Extract}\left(\{(m_i, \sigma_i)\}_{i\in[k]}, pk\right) \end{array} : pk = \mathsf{extractPK}(sk')\right] = \delta$$

4 DSKE from Hash-Based Signature Schemes

Hash-based signature schemes (such as Lamport [Lam79] and Winternitz OTS [BDE$^+$11]) leverage the security of one-way functions to construct digital signatures. In hash-based signature schemes, the private key is often a list that is derived from a succinct seed, and based on the message, the signature reveals "partial" information about the private key. In essence, having a *sufficiently* large number of signatures allows us to obtain enough information for the reconstruction of the private key.

In this section, we provide DSKE constructions based on the hash-based signature schemes, denoted as $\mathtt{DSKE}_{\mathsf{hash}}$. We first give a generic definition to capture the private key leakage of hash-based signature schemes and then provide two DSKE constructions that are based on the Lamport signature scheme and Winternitz OTS (see Appendix A).

Definition 11 (Leakage of Hash-based Signature)**.** *A hash-based signature, Σ_{hash}, consists of the four algorithms* $\mathtt{Setup}$, KeyGen, Sign, *and* Verify *as defined in Definition 7, as well as a leakage algorithm defined as follows:*

- $S/\perp \leftarrow \mathsf{Leak}((m,\sigma),pk)$*: This leakage algorithm is a* deterministic *algorithm that takes as input a message-signature pairs(m,σ) and a public key pk. If $\mathsf{Verify}(pk,m,\sigma)=1$, it outputs a list S, containing a fraction of the private key; Otherwise, it outputs $\perp$.*

We can then propose a hash signature-based DSKE:

Definition 12 (Hash-based DSKE)**.** *A hash-based DSKE scheme, $\mathtt{DSKE}_{\mathsf{hash}}$, consists of the four algorithms,* $\mathtt{Setup}$, KeyGen, Sign, *and* Verify *which are same as a hash-based signature, as well as an* $\mathsf{Extract}$ *algorithm defined as follows:*

- $sk/\perp \leftarrow \mathsf{Extract}(\{(m_i,\sigma_i)\}_{i\in[k]},pk)$*: The extraction algorithm is a deterministic algorithm that receives a set of distinct message-signature pairs $\{m_i,\sigma_i\}_{i\in[k]}$, and the public key pk as inputs. For each (m_i,σ_i), this algorithm runs $S_i \leftarrow \mathsf{Leak}((m_i,\sigma_i),pk)$, and computes $sk' = \bigcup_{i=1}^{k}(S_i)$. The algorithm outputs the private key sk' if $pk = \mathsf{extractPK}(sk')$; otherwise, it outputs $\perp$.*

We subsequently introduce a definition aimed at quantifying the probability that an element of the private key is not revealed even after receiving k hash-based signatures. This definition serves as a tool to compute the probability of a successful output of the private key by running the $\mathsf{Extract}()$ function.

Definition 13. *Given k distinct messages $\{m_i\}_{i\in[k]}$ which are randomly sampled from $\mathcal{M}$, a key pair (pk,sk) generated by $\mathsf{KeyGen}()$ where $sk = (sk_j)_{[\lambda]}$, k signatures $\{\sigma_i\}_{i\in[k]}$ such that each σ_i is a valid signature on m_i under pk, and an element sk_j which is a part of the secret key sk, we define p_k^{leak} as the probability that sk_j is not leaked after running $\mathsf{Leak}()$ on all message-signature pairs $\{(m_i,\sigma_i)\}_{i\in[k]}$.*

$$\mathsf{p}_k^{leak} = \Pr\left[\begin{array}{l} par \leftarrow \mathsf{Setup}(1^\lambda),(pk,sk)\leftarrow \mathsf{KeyGen}(par)\\ m_i \xleftarrow{\$} \mathcal{M},\ \mathit{for}\ i\in[k]\\ \sigma_i \leftarrow \mathsf{Sign}(sk,m_i), S_i \leftarrow \mathsf{Leak}((m_i,\sigma_i),pk)\\ sk_j \in sk \end{array} : sk_j \notin \bigcup_{i=1}^{k} S_i\right]$$

The following theorem shows that the lower bound of δ in any hash-based DSKE scheme can be determined by p_k^{leak}.

Theorem 1 (Extractability with Trusted Generation). *Given a hash-based DSKE scheme with a private key of size λ, if the underlying hash function is modeled as a random oracle, then $\mathtt{DSKE}_{\mathsf{hash}}$ has a (k, δ)-extractable set, where: $\delta \geq 1 - \lambda \cdot \mathsf{p}_k^{leak} - \mathsf{negl}(\lambda)$.*

In this paper, we provide two DSKE constructions based on the Lamport signature scheme and Winternitz OTS, respectively. We will show that, for a Lamport signature scheme, p_k^{leak} is determined by the number of given signatures (i.e., k), $\mathsf{p}_k^{leak} = \frac{1}{2^{k-1}}$ (see Section 4.1); and for a Winternitz OTS, $\mathsf{p}_k^{leak} = \frac{w^k}{(w+1)^k}$, where w are a parameter for the Winternitz OTS (See Appendix A).

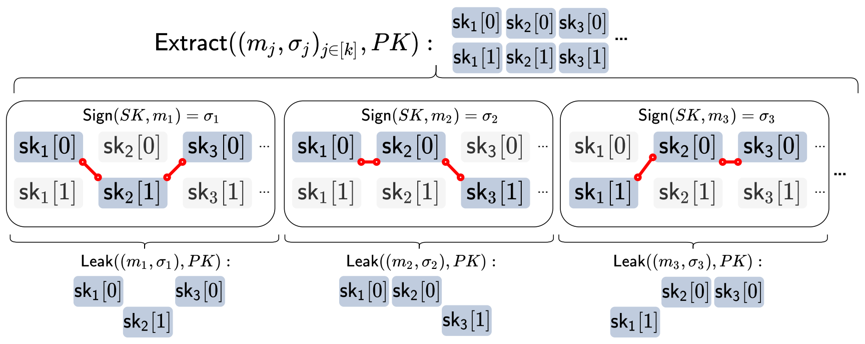
4.1 Lamport Signature-Based Construction

The Lamport signature scheme is parameterized by a preimage-resistant hash function F and a collision-resistant hash function H. The private key SK contains 2λ binary strings uniformly sampled from $\{0,1\}^\lambda$, where λ is the security parameter. The public key PK consists of 2λ binary strings that are evaluations of F on each element in the private key. To form a signature σ on a message m, the signer reveals components of the private key SK, according to the binary representation of $H(m)$ as the signature. To verify the signature, the verifier uses the binary representation of the digest, $H(m)$, to validate if each element contained in the signature is the actual preimage of the public key elements.

Extracting Private Key from Message-Signature Pairs. As shown in Figure 1, the intuition behind the extracting function is that each signature σ_i is a subset of the private key SK. To extract the complete private key, one needs to compute the union of k different signatures $\{\sigma_i\}_{i\in[k]}$. Since the underlying hash function is unpredictable, the probability of successfully extracting the private key depends on the number of distinct message-signature pairs (i.e., k). Based on the above intuition, we propose a hash-based construction, $\mathtt{DSKE}_{\mathsf{lamp}}$, as follows.

Leakage Function for Lamport Signatures. As illustrated in Figure 1, within a Lamport signature scheme, each signature encapsulates half of the information about the privacy key SK. The leakage function is employed to extract the divulged information of the privacy key from the signature.

- $\mathsf{Leak}((m,\sigma), PK)$: Check if $\mathsf{Verify}(PK, m, \sigma) = 1$; otherwise, output $\perp$. Compute $d = H(m) = (d_i)_{i\in[\lambda]}$, and parse $\sigma = (\sigma_i)_{i\in[\lambda]}$. For each $i \in [\lambda], b \in \{0,1\}$, if $d_i = b$, then let $S_i[b] = \sigma_i$ and $S_i[1-b] = \perp$. Output $S = (S_i[b])_{i\in[\lambda], b\in\{0,1\}}$.



Extract$((m_j, \sigma_j)_{j\in[k]}, PK)$:
$sk_1[0]$ $sk_2[0]$ $sk_3[0]$ ...
$sk_1[1]$ $sk_2[1]$ $sk_3[1]$
Sign$(SK, m_1) = \sigma_1$
Sign$(SK, m_2) = \sigma_2$
Sign$(SK, m_3) = \sigma_3$
Leak$((m_1, \sigma_1), PK)$:
Leak$((m_2, \sigma_2), PK)$:
Leak$((m_3, \sigma_3), PK)$:

Figure 1: Example of the algorithm Extract for $\mathsf{DSKE_{lamp}}$. The elements of a signature σ are linked via red lines. In this example, the original private key can be collectively reconstructed from $\{(m_i, \sigma_i)\}_{i\in[3]}$.

We the proceed to define the Lamport-Based DSKE.

Lamport-Based DSKE. A Lamport-based DSKE scheme, $\mathsf{DSKE_{lamp}}$, includes:

- $\mathsf{Setup}(1^\lambda)$: On input the security parameter λ, the algorithm outputs a parameter, *par*, containing a hash function, $F : \{0,1\}^\lambda \to \{0,1\}^\lambda$, chosen from a family of preimage-resistant hash functions, and a hash function, $H : \mathcal{M} \to \{0,1\}^\lambda$, chosen from a family of collision-resistant hash function. The public parameters are implicitly input to all subsequent algorithms.

- $\mathsf{KeyGen}(par)$: For each $i \in [\lambda]$, $b \in \{0,1\}$, sample $sk_i[b] \xleftarrow{\$} \{0,1\}^\lambda$; output the private key, $SK = (sk_i[b])_{i\in[\lambda], b\in\{0,1\}}$, and the public key, $PK = (pk_i[b])_{i\in[\lambda], b\in\{0,1\}}$, where $pk_i[b] = F(sk_i[b])$.

- $\mathsf{Sign}(SK, m)$: Parse $SK = (sk_i[b])_{i\in[\lambda], b\in\{0,1\}}$, compute $d = H(m) = (d_i)_{i\in[\lambda]}$, and output $\sigma = (sk_i[d_i])_{i\in[\lambda]}$.

- $\mathsf{Verify}(PK, m, \sigma)$: Parse PK, σ, and compute $d = H(m) = (d_i)_{i\in[\lambda]}$. For all $i \in [\lambda]$, if $F(\sigma_i) \neq pk_i[d_i]$, return 0. Otherwise, return 1.

- $\mathsf{Extract}(\{m_j, \sigma_j\}_{j\in[k]}, PK)$: For each message-signature pair (m_j, σ_j), compute $d_j = H(m_j) = (d_{ji})_{i\in[\lambda]}$, and parse $\sigma_j = (\sigma_{ji})_{i\in[\lambda]}$. For each $j \in [k], i \in [\lambda], b \in \{0,1\}$, if $\exists d_{ji} = b$, then let $sk_i[b] = \sigma_{ji}$. Set $SK = (sk_i[b])_{i\in[\lambda], b\in\{0,1\}}$ and parse $PK = (pk_i[b])_{i\in[\lambda], b\in\{0,1\}}$. If $\forall i \in [\lambda]$ and $b \in \{0,1\}$, $pk_i[b] = pk_i[d_i] = F(\sigma_i) = F(sk_i[b])$, then output SK. Otherwise, the algorithm outputs $\perp$.

$\mathsf{DSKE_{lamp}}$ satisfies unforgeability and extractability with trusted generation depending on the number of distinct message-signature pairs. We refer readers to our detailed proofs in Appendix B.1.

Theorem 2 (Existential Unforgeability)**.** $\mathtt{DSKE}_{\mathsf{lamp}}$ *is existentially unforgeable under a* 1*-time adaptive chosen-message attack, that is,*

$$\Pr[\mathsf{SignExp}^{1}_{\mathcal{A},\mathit{DSKE}_{\mathsf{lamp}}}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$$

Lemma 1. *In a Lamport-based DSKE scheme, if $H(\cdot)$ is modeled as a random oracle, the probability that an element of the private key will not be leaked after receiving k distinct signatures is $\mathsf{p}_k^{leak} = \frac{1}{2^{k-1}}$.*

By combining Theorem 1 and Lemma 1, we can deduce Theorem 3.

Theorem 3 ($\mathtt{DSKE}_{\mathsf{lamp}}$ Extractability with Trusted Generation)**.** *If $H(\cdot)$ is modeled as a random oracle, then $\mathit{DSKE}_{\mathsf{lamp}}$ has a (k, δ)-extractable set, where: $\delta \geq 1 - \dfrac{\lambda}{2^{k-1}} - \mathsf{negl}(\lambda)$.*

Limitations of $\mathtt{DSKE}_{\mathsf{hash}}$. Although the scheme $\mathtt{DSKE}_{\mathsf{hash}}$ satisfies our definition of a signature scheme with key extraction, it has several inherent weaknesses: (1) The key and signature sizes are linear with the security parameter λ, making it inefficient in practice; (2) $\mathsf{Extract}$ may not always output the secret key and depends on the size of the extractable set k [2]; (3) $\mathtt{DSKE}_{\mathsf{hash}}$ does not satisfy the extractability with an untrusted generation, as a (malicious) signer can manipulate the hash results of messages. For instance, within $\mathtt{DSKE}_{\mathsf{lamp}}$, a signer may selectively choose messages to manipulate the hash results: the first bit of the hash results $(H(m_1), H(m_2), \dots)$ consistently remains 0, thus ensuring that the private key will never get extracted.

5 DSKE from Polynomial Commitment Schemes

In this section, we construct a DSKE scheme $\mathtt{DSKE}_{\mathsf{poly}}$ from a polynomial commitment scheme Π without the drawbacks of $\mathtt{DSKE}_{\mathsf{hash}}$. The idea is to use the polynomial $f(X)$ of degree d as the private key with the corresponding public key being the polynomial commitment. Then, the signature on a message m is the evaluation of $f(X)$ at point $x = H(m)$, where H is a collision-resistant hash function. For key extraction, we employ polynomial interpolation: any set of $d + 1$ valid message-signature pairs (m_i, σ_i) can reconstruct $f(X)$.

Definition 14 (Polynomial Commitment-based DSKE)**.** *A polynomial commitment-based DSKE scheme, $\mathtt{DSKE}_{\mathsf{poly}}$, consists of:*

[2]This drawback can be circumvented by requiring signers to add an extra nonce to k different messages so that the set of these signatures guarantees the existence of the extractable set.

- $\mathsf{Setup}(1^\lambda, d)$*: On input the security parameter λ and degree $d \in \mathbb{N}$, it runs* $\Pi.\mathsf{Setup}(1^\lambda, d)$ *to obtain* (ck, vk)*, which allows a signer to commit to polynomials in* $\mathbb{F}^{\leq d}(X)$*. It samples a collision-resistant hash function* $H : \mathcal{M} \to \mathbb{F}$*. The public parameters,* par*, contain* ck, vk, d*, and the specification of* H*.*

- $\mathsf{KeyGen}(par)$*: It samples* $f(X) \xleftarrow{\$} \mathbb{F}^{\leq d}(X)$ *as a* d*-degree polynomial and computes* $\Pi.\mathsf{Com}(ck, f(X)) \to C_f$*. Let* $sk = f(X)$*,* $pk = C_f$ [3]*.*

- $\mathsf{Sign}(sk, m)$*: It parses* $sk = f(X)$*, computes* $x = H(m)$*, and runs* $\Pi.\mathsf{Open}(ck, C_f, x, f(X)) \to (\pi, y)$*. It outputs the signature* $\sigma = (\pi, y)$*.*

- $\mathsf{Verify}(pk, m, \sigma)$*: It parses* $pk = C_f$ *and* $\sigma = (\pi, y)$*, computes* $x = H(m)$*, and outputs* $\Pi.\mathsf{Check}(vk, C_f, x, y, \pi) \in \{0, 1\}$*.*

- $\mathsf{Extract}(\{(m_i, \sigma_i)\}_{i \in [k]}, pk)$*: If the number of distinct* m_i *is less than* d*, or* $\mathsf{Verify}(pk, m_i, \sigma_i) = 0$ *for some* $i \in [k]$*, then return* $\perp$*. Otherwise, it computes* $x_i = H(m_i)$ *and parses* $\sigma_i = (\pi_i, y_i)$*, for* $i \in [k]$*, and outputs* $f'(X) \leftarrow \mathsf{Interpolate}(\{x_i, y_i\}_{i \in [d+1]})$

In the following, we show that $\mathtt{DSKE}_{\mathsf{poly}}$ can achieve existential unforgeability, extractability with both trusted and untrust generation. Moreover, in this construction, $\mathsf{extractPK}(\cdot) = \Pi.\mathsf{Com}(\cdot)$.

Theorem 4 (Existential Unforgeability)**.** *Assuming the security of underlying polynomial commitment scheme* Π*, and that* H *is a hash function modeled as random oracle, the DSKE scheme* $\mathtt{DSKE}_{\mathsf{poly}}$ *is* existentially unforgeable *under* d*-times adaptive chosen-message attack. That is,* $\Pr[\mathsf{SignExp}^{d}_{\mathcal{A}, DSKE_{\mathsf{poly}}}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$*.*

We refer readers to our detailed proofs of Theorem 4 in Appendix B.2.

Theorem 5 (Extractability with Trusted Generation)**.** *Assuming the security of underlying polynomial commitment scheme* Π*, the DSKE scheme* $\mathtt{DSKE}_{\mathsf{poly}}$ *has a* $(k, 1 - \mathsf{negl}(\lambda))$*-extractable set with trusted generation for any* $k \geq d + 1$*. That is,*

$$\Pr\left[\begin{array}{l} par \leftarrow \mathsf{Setup}(1^\lambda), (pk, sk) \leftarrow \mathsf{KeyGen}(par) \\ m_i \leftarrow \mathcal{M}, s.t.\ m_i \neq m_j,\ for\ i, j \in [k], i \neq j \\ \sigma_i \leftarrow \mathsf{Sign}(sk, m_i),\ for\ i \in [k] \\ sk' \leftarrow \mathsf{Extract}\left(\{(m_i, \sigma_i)\}_{i \in [k]}, pk\right) \end{array} : pk = \mathsf{extractPK}(sk') \right] = 1 - \mathsf{negl}(\lambda)$$

[3]For applications that require deniability, the signer can flexibly choose to commit to any polynomial of degree $\ell \in [1, d]$ and has no incentive to commit to a polynomial of degree larger than ℓ, as that would cost them the deniability property, as we discuss in the following sections.

Proof. The proof follows from two facts. First, since the key generation is trusted, the degree of the polynomial must be less than d and the signer does not commit to polynomials of degree bigger than d. Second, from the *evaluation binding property* and since the points (x_i, y_i) correspond to valid signatures, we know that $y_i = f(x_i)$, for some polynomial $f(X) \in \mathbb{F}^{\leq d}(X)$ and for all $i \in [k]$, except with negligible probability. Due to the uniqueness of polynomial interpolation, we know that any $d+1$ distinct points (x_i, y_i) define a unique polynomial $\phi(X)$ of degree at most d, hence $\phi(X)$ must be the same as $f(X)$, hence $sk = sk'$ with probability $1 - \mathsf{negl}(\lambda)$. □

Theorem 6 (Extractability with Untrusted Generation)**.** *Assuming the security of underlying polynomial commitment scheme* Π*, the DSKE scheme* $\mathsf{DSKE}_{\mathsf{poly}}$ *has a* $(k, 1 - \mathsf{negl}(\lambda))$*-extractable set with untrusted generation for any* $k \geq d+1$*. That is,*

$$\Pr\left[\begin{array}{l} par \leftarrow \mathsf{Setup}(1^\lambda), (pk, \{(m_i, \sigma_i)\}_{i\in[k]}) \leftarrow \mathcal{A}(1^\lambda, par), \\ s.t.\ \mathsf{Verify}(pk, m_i, \sigma_i) = 1, \\ m_i \neq m_j,\ for\ i \neq j, \\ sk' \leftarrow \mathsf{Extract}\left(\{(m_i, \sigma_i)\}_{i\in[k]}, pk\right) \end{array} : pk = \mathsf{extractPK}(sk')\right] = 1 - \mathsf{negl}(\lambda)$$

Proof. Consider an adversary $\mathcal{A}$ capable of submitting a public key pk and k signatures $\sigma_1, \ldots, \sigma_k$ on distinct messages $m_1, \ldots, m_k$ such that $\mathsf{Verify}(pk, m_i, \sigma_i) = 1$ for $i \in [1, k]$ and $k \geq d+1$. If the private key $f(X)$ cannot be extracted from their corresponding signatures, we encounter two possible scenarios:

1. The k messages $m_1, \ldots, m_k$ generate fewer than $d+1$ inputs for $f(X)$, implying the existence of some m_i and m_j ($i \neq j$, $i, j \in [k]$) for which $H(m_i) = H(m_j)$, which breaks the collision-resistance property of H.
2. $\mathcal{A}$ directly breaks the interpolation binding property of PCS: Parse $\sigma_i = (\pi_i, y_i)$ and set $x_i = H(m_i)$ for $i \in [1, d+1]$, we can submit $(pk, \{x_i, y_i, \pi_i\}_{i\in[1,d+1]})$ which immediately breaks interpolation binding of the polynomial commitment scheme Π.

Both scenarios occur with negligible probability, leading to $\delta = 1 - \mathsf{negl}(\lambda)$. □

5.1 DSKE from KZG Polynomial Commitment

KZG Polynomial Commitment Scheme. We now revisit the KZG scheme [KZG10] as a concrete polynomial commitment construction. It works over a bilinear pairing group $\mathbb{G} = \langle e, G, G_t \rangle$, where G is a group of prime order p, e is a symmetric pairing $e : G \times G \to G_t$, and g and $\hat{h}$ are generators of G.

- $(\mathbb{G}, \mathsf{ck}, \mathsf{vk}) \leftarrow \mathsf{Setup}(1^\lambda, d)$: The algorithm outputs a representation of the bilinear group $\mathbb{G}$, commitment key $ck = \{g, g^\alpha, \dots, g^{\alpha^d}\}$, and verification key $vk = \hat{h}^\alpha$, for an $\alpha \in \mathbb{Z}_p$.

- $C_f \leftarrow \mathsf{Com}(ck, f(X))$: The algorithm computes $C_f = g^{f(\alpha)}$ using ck and outputs C_f.

- $(\pi, y) \leftarrow \mathsf{Open}(ck, C_f, x, f(X))$: The algorithm computes $y = f(x)$ and the quotient polynomial $q(X) = \frac{f(X)-y}{X-x}$, and outputs y and $\pi = C_q = \mathsf{Com}(ck, q(X))$.

- $0/1 \leftarrow \mathsf{Check}(vk, C_f, x, y, \pi)$: The algorithm outputs 1 if $e(C_f \cdot g^{-y}, \hat{h}) = e(C_q, \hat{h}^\alpha \cdot \hat{h}^{-x})$, and 0 otherwise.

Kate *et. al.* [KZG10] proved that KZG scheme satisfies *correctness*, *computational hiding*, *evaluation binding*, *polynomial binding* properties, provided the DL and d-SDH assumptions hold in $\mathbb{G}$ [KZG10]. Recently, Abraham *et. al.* [AJM$^+$23] proved that KZG PCS satisfies *interpolation binding* property assuming the hardness of the Strong Diffie-Hellman problem in the Algebraic Group Model.

KZG-based DKSE Scheme. We now show $\mathtt{DSKE}_{\mathsf{poly}}$, a concrete construction of $\mathtt{DSKE}_{\mathsf{poly}}$ from the Π_{kzg} polynomial commitment scheme [KZG10].

- $\mathsf{Setup}(1^\lambda, d)$: Run $\Pi_{\mathsf{kzg}}.\mathsf{Setup}(1^\lambda, d)$ to obtain the commitment key, $ck = \{g, g^\alpha, \dots, g^{\alpha^d}\}$, and the verification key, $vk = \hat{h}^\alpha \in G$. It samples a collision-resistant hash function $H : \mathcal{M} \rightarrow \mathbb{Z}_p$ The algorithm returns the public parameters, par, containing ck, vk, d, and the specification of H.

- $\mathsf{KeyGen}(1^\lambda)$: Sample $f(X) \xleftarrow{\$} \mathbb{F}^d(X)$. The algorithm returns the public key, $pk = C_f = \Pi_{\mathsf{kzg}}.\mathsf{Com}(ck, f(X)) = g^{f(\alpha)} \in G$ and the secret key, $sk = f(X)$.

- $\mathsf{Sign}(sk, m)$: Parse $sk = f(X)$, compute $x = H(m)$, and output the signature $\sigma = (\pi, y) = \Pi_{\mathsf{kzg}}.\mathsf{Open}(ck, C_f, x, f(X)) \in G \times \mathbb{Z}_p$.

- $\mathsf{Verify}(pk, m, \sigma)$: Parse $pk = C_f$ and $\sigma = (\pi, y)$, compute $x = H(m)$, and output $\Pi_{\mathsf{kzg}}.\mathsf{Check}(vk, C_f, x, y, \pi) \in \{0, 1\}$.

- $\mathsf{Extract}(\{(m_i, \sigma_i)\}_{i \in [k]}, pk)$: As in the general construction, check the validity of σ_i, for $i \in [k]$, interpolate $\phi(X)$ from points (x_i, y_i) output $sk' = \phi(X) \in \mathbb{F}^d(X)$.

$\mathtt{DSKE}_{\mathsf{kzg}}$ can satisfy existential unforgeability (see Theorem 4), and the extractability with trusted generation (see Theorem 5) as well as untrusted generation (see Theorem 6).

6 Applications

In this section, we present two concrete applications of DSKE.

6.1 Non-Attributable Email

Non-Attributable Email. The notion of non-attributable emails [SPG21] addresses the inherent drawbacks of DKIM by deploying a forward-forgeable signature (FFS) scheme. An FFS scheme consists of the standard $\mathsf{KeyGen}(1^\lambda)$, $\mathsf{Sign}(sk, m)$, and $\mathsf{Verify}(pk, m, \sigma)$ algorithms, and additionally an $\mathsf{Expire}(sk, \mathcal{T})$ algorithm, that generates expiry information η for a private key sk, possibly using additional information $\mathcal{T}$, and a $\mathsf{Forge}(\eta, m)$ algorithm, that, given the expiry information of a key, outputs a signature on a message m. The scheme satisfies the standard *correctness* and *unforgeability* properties, and the *forgeability on expiry* property.

Definition 15 (Forgeability on Expiry [SPG21]). *A digital signature scheme satisfies the* forgeability on expiry *property if no PPT adversary can distinguish a signature created with private key sk from a signature created with the expiry information η of sk. Formally, for any $m \in \mathcal{M}$ and any PPT distinguisher $\mathcal{D}$, there is a negligible function* $\mathsf{negl}(\cdot)$*, such that for all λ,*

$$\Pr\left[\begin{array}{l}(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda), \sigma_0 \leftarrow \mathsf{Sign}(sk, m)\\ \eta \leftarrow \mathsf{Expire}(sk, \mathcal{T}), \sigma_1 \leftarrow \mathsf{Forge}(\eta, m)\\ b \xleftarrow{\$} \{0,1\}, b' \leftarrow \mathcal{D}(\sigma_b, \eta, m, pk)\end{array} : b = b'\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

The authors in [SPG21] provide two constructions, KeyForge and Timeforge. Both satisfy the FFS properties by requiring signers either to publish old private keys or to issue signed updates to a publicly verifiable timekeeping service, respectively. KeyForge is an FFS scheme where signatures expire after a predefined delay Δ. The expiry information is the private key itself, which has to be published by the server every Δ time. KeyForge uses a hierarchical tree structure with four levels to manage private keys, where each level corresponds to distinct timespans, namely, years, months, days, and minutes. Each leaf represents a private key of a unique 15-minute time chunk. This structure is realized by a hierarchical identity-based signature scheme (HIBS), which allows any node to derive the private keys of its children. The underlying HIBS enables email servers to keep the expiration information succinct. Instead of storing all private keys for a day as the expiration information, it is enough to store the private key that corresponds to the node that encodes that day on the third level.

GroupForge from DSKE. In the following, we show how a DSKE scheme, such as $\mathtt{DSKE}_{\mathsf{poly}}$, can be used in a non-attributable email protocol [SPG21]. DSKE removes the requirement for email servers to periodically publish

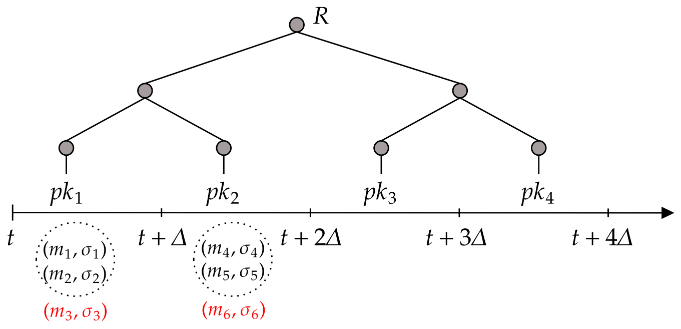


Figure 2: An example of GroupForge for $h_T = 2$ and between the interval $[t, t+2^{h_T} \cdot \Delta]$. In this example, $\{(m_i, \sigma_i)\}_{i\in[2]}$ can potentially act as the deniable group (see Section 8 for detailed discussions) for the message-signature pair (m_3, σ_3), and $\{(m_i, \sigma_i)\}_{i\in\{4,5\}}$ can act as the deniable group for the message-signature pair (m_6, σ_6).

expiration information. Looking ahead, we introduce GroupForge, an FFS scheme that builds on top of a DSKE and achieves the same properties as KeyForge, while it does not require the email servers to publish *any* expiry information, i.e., algorithm $\mathsf{Expire}()$ requires only information already published by the servers.

GroupForge uses a DSKE scheme Σ_{DSKE} with extractable sets of size k, a collision-resistant hash function $H()$, and a clock $\mathsf{Time}(\cdot)$ that returns the current time. GroupForge works as follows.

- $\mathsf{KeyGen}(1^\lambda, \Delta, h_T, t)$: On input the security parameter λ, the length Δ, the height of the Merkle tree $h_T \in \mathbb{N}$, and the starting time t, it invokes $\Sigma_{\mathsf{DSKE}}.\mathsf{Setup}(1^\lambda)$, and generates 2^{h_T} key pairs $\{(pk_i, sk_i)\}_{i\in[2^{h_T}]}$ using the algorithm $\Sigma_{\mathsf{DSKE}}.\mathsf{KeyGen}(par)$. The algorithm then uses the hash function $H()$ to construct a Merkle tree from $\{pk_i\}_{i\in[2^{h_T}]}$. Let R be the root of this Merkle tree. Then it sets $PK = (R, \Delta, h_T, t)$ and $SK = (R, \Delta, h_T, t, \{sk_i\}_{i\in[2^{h_T}]})$, and outputs (SK, PK).

- $\mathsf{Sign}(SK, m)$: On input the key, $SK = (R, \Delta, h_T,\ t, \{sk_i\}_{i\in[2^{h_T}]})$, it obtains the current time $t' \leftarrow \mathsf{Time}()$, and computes $i = \lfloor (t'-t)/\Delta \rfloor$ (the current time chunk $\lfloor (t'-t)/\Delta \rfloor$). It invokes $s = \Sigma_{\mathsf{DSKE}}.\mathsf{Sign}(sk_i, m)$, the authenticity proof proof_i for the path from leaf_i to the root R, and outputs the signature $\sigma = (s, pk_i, \mathsf{leaf}_i, \mathsf{proof}_i)$.

- $\mathsf{Verify}(PK, m, \sigma)$: On input the public key PK, the message m, and the signature σ, it parses $PK = (R, \Delta, h_T, t)$, and $\sigma = (s, pk_i,\ \mathsf{leaf}_i, \mathsf{proof}_i)$, obtains the current time $t' \leftarrow \mathsf{Time}()$, and computes $i' = \lfloor (t'-t)/\Delta \rfloor$. If $i \neq i'$ (i.e., messages arrived late), then returns 0. It validates proof_i starting from leaf_i; if invalid, it returns 0. Otherwise, returns $\Sigma_{\mathsf{DSKE}}.\mathsf{Verify}(pk_i, m, s)$.

- $\mathsf{Expire}(PK, \{(m_j, \sigma_j)\}_{j\in[k]})$: On input the public key PK and k message-signature pairs $\{(m_j, \sigma_j)\}_{j\in[k]}$, the algorithm checks that the messages are pairwise different, i.e., for all $j_1, j_2 \in [k]$, $m_{j_1} \neq m_{j_2}$, that all signatures are valid, i.e., for all $j \in [k]$, $\mathsf{Verify}(PK, m_j, \sigma_j) = 1$, and that all signatures are created in the same time chunk, i.e., there exists $i \in [2^{h_T}]$ such that for all $j \in [k]$, $\sigma_j = (s_j, pk_i, \mathsf{leaf}_i, \mathsf{proof}_i)$. If the conditions do not hold, it returns $\perp$. Otherwise, it calls $\Sigma_{\mathsf{DSKE}}.\mathsf{Extract}(\{(m_j, \sigma_j)\}_{j\in[k]}, pk_i)$. If this call returns $\perp$, then $\mathsf{Expire}()$ also returns $\perp$. If it returns an extracted key sk_i, then $\mathsf{Expire}()$ returns the expiry information $\mathsf{info} = (\mathsf{leaf}_i, \mathsf{proof}_i, sk_i)$.

- $\mathsf{Forge}(PK, m, \mathsf{info})$: On input the public key PK, a message m, and the expiry information $\mathsf{info} = (\mathsf{leaf}_i, \mathsf{proof}_i, sk_i)$, for some $i \in [2^{h_T}]$, the algorithm computes $s = \Sigma_{\mathsf{DSKE}}.\mathsf{Sign}(sk_i, m)$ and returns the signature $\sigma = (s, pk_i, \mathsf{leaf}_i, \mathsf{proof}_i)$, where $pk_i = \mathsf{extractPK}(sk_i)$ is the public key of leaf_i.

Figure 2 provides a pictorial example of how GroupForge works. GroupForge satisfies the *correctness*, *unforgeability*, and *forgeability on expiry* properties of an FFS scheme (see Appendix B.3).

Remark. A close notion to FSS is the concept of short-lived signatures [ABC22], where the signature becomes non-attributable after some time without requiring further actions from the signer like releasing expiry information. We later in Section 8 discuss that GroupForge can also work as a short-lived signature.

6.2 Rate-Limiting Nullifier

Rate Limiting Nullifier. A Rate-Limiting Nullifier (RLN) [Pri24] is a scheme aimed at restricting the number of user actions, thereby facilitating a robust mechanism for spam prevention. This approach has been extensively discussed in a range of real-world applications [Bla21b, BB21]. Generally, RLN achieves spam deterrence by either increasing the difficulty of duplicating identities or disclosing of a user's private key once a certain number of actions are exceeded. In the following, we propose a RLN construction that is directly derived from our $\mathtt{DSKE}_{\mathsf{poly}}$ scheme. The formal study and construction of secure RLN schemes has, to the best of our knowledge, not appeared in academic literature before.

RLN from $\mathtt{DSKE}_{\mathsf{poly}}$. A RLN scheme typically consists of four parts, *setup*, *registration*, *interaction*, and *slashing*, run between two types of participants, a *user* $\mathcal{U}$ and a *server* $\mathcal{T}$. Our $\mathtt{DSKE}_{\mathsf{poly}}$-based RLN works as follows:

- $\mathsf{Setup}(1^\lambda, d, \mathcal{U}, \mathcal{T})$: On input the security parameter λ and degree $d \in \mathbb{N}$, it runs $\mathtt{DSKE}_{\mathsf{poly}}.\mathsf{Setup}(1^\lambda, d)$ to obtain (ck, vk), which allows a signer to commit to polynomials in $\mathbb{F}^{\leq d}(X)$. It samples a collision-resistant hash function $h : \mathcal{M} \to \mathbb{F}$. The public parameters par contain ck, vk, d, and h. The parameters are publicly known for the user $\mathcal{U}$ and the server $\mathcal{T}$.

- $\mathsf{Registration}(1^\lambda, \mathcal{U}, \mathcal{T})$: The user $\mathcal{U}$ runs $\mathtt{DSKE}_{\mathsf{poly}}.\mathsf{KeyGen}(1^\lambda, d)$ to obtain the private key $sk_{\mathcal{U}} = f(X)$ and the public key $pk_{\mathcal{U}} = C_f$. The public key $pk_{\mathcal{U}}$ is registered and stored on the server $\mathcal{T}$. $\mathcal{T}$ initializes a counter $\mathsf{MsgNum}_{\mathcal{U}} = 0$ to record the number of messages signed by $\mathcal{U}$.

- $\mathsf{Interaction}(m, \mathcal{U}, \mathcal{T})$: The interaction algorithm is run between $\mathcal{U}$ and $\mathcal{T}$ in order to sign and verify a message m. User $\mathcal{U}$ parses $sk = f(X)$, computes $x = h(m)$ and the signature $\sigma = \mathtt{DSKE}_{\mathsf{poly}}.\mathsf{Open}(ck, C_f, x, f(X)) = (\pi, y)$, and sends σ to the server $\mathcal{T}$. Upon receiving σ, server $\mathcal{T}$ parses the user's $pk_{\mathcal{U}} = C_f$ and $\sigma = (\pi, y)$, computes $x = h(m)$, and outputs $\mathtt{DSKE}_{\mathsf{poly}}.\mathsf{Check}(vk, C_f, x, y, \pi) \in \{0, 1\}$. If $\mathtt{DSKE}_{\mathsf{poly}}.\mathsf{Check}(vk, C_f, x, y, \pi) = 1$, then $\mathcal{T}$ increases the counter $\mathsf{MsgNum}_{\mathcal{U}}$ and stores the message-signature pair (m, σ).

- $\mathsf{Slashing}(\{(m_i, \sigma_i)\}_{i \in [\mathsf{MsgNum}]}, pk_{\mathcal{U}}, \mathcal{T})$: If server $\mathcal{T}$ has received more than d message-signature pairs from the user $\mathcal{U}$, i.e., $\mathsf{MsgNum} > d$, then $\mathcal{T}$ runs the algorithm $\mathsf{Extract}(\{(m_i, \sigma_i)\}_{i \in [\mathsf{MsgNum}]}, pk_{\mathcal{U}})$ of $\mathtt{DSKE}_{\mathsf{kzg}}$ to extract $sk_{\mathcal{U}}$. Subsequently, $\mathcal{T}$ publishes $sk_{\mathcal{U}}$ and removes $\mathcal{U}$ from the system.

Spam Prevention in $\mathtt{DSKE}_{\mathsf{poly}}$-based RLN. The key extraction property of a $\mathtt{DSKE}_{\mathsf{ploy}}$ (see Theorem 5) empowers the server to extract the private key of a user who issues more than d messages with their signatures. Here, d serves as a predefined parameter, influencing the polynomial's degree that the user can select. Consequently, in a $\mathtt{DSKE}_{\mathsf{poly}}$-based RLN, a user is prevented from generating excessive spam (exceeding d messages) because their private key would be publicly disclosed, leading to their expulsion from the system.

Using RLN in Anonymous Setting. As discussed in existing works [Bla21b, Res21, Bla21a, BB21], we can utilize non-interactive zero-knowledge proofs to hide the public key $pk_{\mathcal{U}}$ of a user $\mathcal{U}$, thereby achieving anonymity for an RLN scheme. Nevertheless, a significant challenge persists in obtaining the extractable set, which consists of signatures from the same user. The full anonymity provided by zero-knowledge proofs leaks nothing about the signer, preventing anyone to obtain this set. A feasible approach might be to weaken the full anonymity to the linkable anonymity by including a linkable tag into the signed message. It allows the server to link the signatures and track the number of messages originating from the same user (without revealing the public key). Incorporating these techniques for privacy protection into our $\mathtt{DSKE}_{\mathsf{poly}}$-enhanced RLN represents a promising direction for future work.

Table 1: Performance of $\mathsf{DSKE}_{\mathsf{lamp}}$ and $\mathsf{DSKE}_{\mathsf{kzg}}$. Note that the probability of extracting the key in $\mathsf{DSKE}_{\mathsf{lamp}}$ is overwhelming in k, so the actual size of the extractable group can be smaller than $k = 64$.

Scheme	Group Size	Key Generation	Signing	Verification	Extraction
$\mathsf{DSKE}_{\mathsf{lamp}}$(SHA256)	$k = 64$	$451.583\mu s$	$17.625\mu s$	$160.458\mu s$	$45.083\mu s$
$\mathsf{DSKE}_{\mathsf{kzg}}$(BLS12-377)	$k = 16$	$2.795ms$	$2.425ms$	$4.949ms$	$1.785ms$
	$k = 32$	$2.825ms$	$2.523ms$	$6.123ms$	$7.813ms$
	$k = 64$	$2.852ms$	$2.756ms$	$7.202ms$	$35.341ms$
	$k = 128$	$3.624ms$	$3.404ms$	$8.045ms$	$166.433ms$

Table 2: Performance of GroupForge Constructions with $\Delta = 0.5$ hour.

Scheme	Parameters: Tree height (Duration)	Key Generation	Signing	Verification
GroupForge$_{\mathsf{hash}}$ (SHA256)	$h_T = 14$ (0.93 years)	$7.407s$	$21.791\mu s$	$176.541\mu s$
	$h_T = 15$ (1.87 years)	$14.815s$	$34.708\mu s$	$177.416\mu s$
	$h_T = 16$ (3.74 years)	$29.631s$	$41.291\mu s$	$178.083\mu s$
	$h_T = 17$ (7.48 years)	$59.262s$	$45.916\mu s$	$179.374\mu s$
GroupForge$_{\mathsf{kzg}}$ (SHA256, BLS12-377, $k = 32$)	$h_T = 14$ (0.93 years)	$46.294s$	$1.554ms$	$6.139ms$
	$h_T = 15$ (1.87 years)	$92.588s$	$1.591ms$	$6.140ms$
	$h_T = 16$ (3.74 years)	$185.175s$	$1.611ms$	$6.141ms$
	$h_T = 17$ (7.48 years)	$370.350s$	$1.722ms$	$6.143ms$

7 Evaluation

In this section, we evaluate the performance of hash-based and polynomial commitment-based DSKE, as well as the performance of GroupForge.

Testbed. We evaluate our schemes on a macOS Monterey machine with an Apple M1 chip (8-core, 3.2 GHz), 8 GB of RAM.

DSKE Constructions. We have implemented prototypes of our proposed constructions of $\mathsf{DSKE}_{\mathsf{lamp}}$ and $\mathsf{DSKE}_{\mathsf{poly}}$ in Rust. For our $\mathsf{DSKE}_{\mathsf{lamp}}$, we use the SHA-256 implementation. For $\mathsf{DSKE}_{\mathsf{poly}}$, we used the KZG polynomial commitment implementation using the curve BLS12-377 from `arkworks` library [4].

Table 1 provides a comprehensive overview of the performance metrics for both $\mathsf{DSKE}_{\mathsf{lamp}}$ and $\mathsf{DSKE}_{\mathsf{poly}}$. Remarkably, the execution time of functions within $\mathsf{DSKE}_{\mathsf{lamp}}$ consistently remains below $1ms$, indicating efficient processing across various $\mathsf{DSKE}_{\mathsf{lamp}}$ functions. The extraction time of $\mathsf{DSKE}_{\mathsf{poly}}$ appears as approximately a quadratic function of the group size k. This is because the complexity of $\phi(X)$ in $\mathsf{DSKE}_{\mathsf{poly}}$ increases quadratically as k grows.

GroupForge Constructions. We also implement two GroupForge constructions based on $\mathsf{DSKE}_{\mathsf{lamp}}$ and $\mathsf{DSKE}_{\mathsf{poly}}$ respectively. We leverage the Merkle tree library using SHA-256 from `arkworks` [5] to construct our GroupForge. We implement GroupForge$_{\mathsf{hash}}$ with $\mathsf{DSKE}_{\mathsf{lamp}}$, and

[4]https://github.com/arkworks-rs/poly-commit

[5]https://github.com/arkworks-rs/crypto-primitives

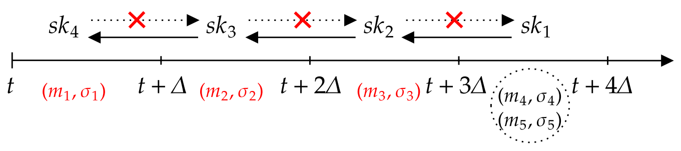


Figure 3: An example of GroupForge using FSS. The group $\{(m_i, \sigma_i)\}_{i \in \{4,5\}}$ can act as a deniable group that can reconstruct the private key to deny all message-signature pairs in the past, $\{(m_i, \sigma_i)\}_{i \in [3]}$.

GroupForge$_{\mathsf{poly}}$ with $\mathtt{DSKE}_{\mathsf{poly}}$ using $\mathsf{BLS12\text{-}377}$ curve and the group size $k = 32$. Table 2 summarizes the performance of GroupForge constructions. We evaluate our constructions with Merkle tree heights of 14, 15, 16, and 17, which correspond to a duration of 0.93, 1.87, 3.74 and 7.48 years respectively (given a time chunk of $\Delta = 0.5$ hour). For both GroupForge$_{\mathsf{hash}}$ and GroupForge$_{\mathsf{poly}}$, the most expensive computation is the key generation. This is because we need to generate 2^{h_T} key pairs, where h_T is the Merkle tree height.

8 Discussion

In this part, we discuss relevant properties and examine potential future directions and improvements for our constructions.

GroupForge with Forward-secure Signature Schemes. Forward-secure signature schemes (FSS) [6] [BM99] allow signers to derive future keys from past keys while preventing users from deriving past keys from future keys. This property of FSS can be combined with GroupForge to demonstrate that if there is a deniable group at some point, the same group can be used as a deniable proof for all message-signature pairs before that point. In particular, to achieve this property, the signer can produce several FSS key pairs from a master private key, and use them in reverse order in the leaves of the Merkle hash tree. For instance, a private key sk_i used in the interval $[t_i, t_i + \Delta]$ can produce all previous private keys sk_j used before t_i. Hence, it is not difficult to see that, if a deniable group exists at some time t', the same group can derive all private keys before t'. Figure 3 gives a high-level example of how FSS can help improve GroupForge.

GroupForge as a Short-lived Signature. There is a subtle difference between forward-forgeable signature (FFS) scheme and a short-lived signature scheme. While FFS provides non-attributability by selective release of some *expiry* information, short-lived signatures *automatically* become non-attributable after some time without further action. GroupForge, proposed in Section 6.1, can essentially offer both types of functionality. In other

[6]FSS is not to be confused with forward-forgeable signatures

words, as the deniability property of GroupForge relies on the *number* of generated signatures, it essentially can be considered a short-lived signature in scenarios where the signer is supposed to generate a *certain* number of signatures, achieving deniability for free without requiring any trusted service or computing costly VDFs. As mentioned, GroupForge can also be modeled as a forward-forgeable signature, achieving forgeability without constantly publishing key materials that are problematic in practice due to unreliable distribution [SPG21].

Trusted Setup. The KZG scheme used in $\mathtt{DSKE}_{\mathsf{poly}}$ requires a trusted setup to generate the public parameters. Although the scheme can leverage a trusted third party to run the setup, such a reliance on a trusted party is typically not welcome in various settings. To remove this trusted assumption, practitioners and the academic community have developed practical solutions to securely generate these parameters, knows as ceremony [NRBB22]. Also, one may deploy polynomial commitment schemes that do not require a trusted setup, such as the one constructed from Bulletproofs [BBB$^+$18].

Time-Agnostic Forgeability. Although not explicitly mentioned, making a synchronous assumption on the communication is required to ensure parties (i.e., group of size k) receive their signatures in the respective time chuck. Moreover, the proper operation of the system relies on parties having access to a local clock that advances at the same pace. The original work of [SPG21] requires the stronger assumption of a shared global clock. However, we observe that the forgeability aspect of the DSKE is generally independent of the notion of time as it comes from the availability of a sufficient number of signatures for key extraction. This essentially opens up the door to use DSKE for adding forgeability property in settings where reliance on time is not desirable.

Compacting the Keys and Signatures. We can also consider extending $\mathtt{DSKE}_{\mathsf{poly}}$ to *multi-signature schemes*, where a group of individuals, each holding their own key pairs, collectively sign a message and the resulting signature is verifiable under the respective set of public keys. In recent years, there has been a surge of interest in multi-signature schemes that support an aggregation property for the public key and the signature, aiming to reduce the size of blockchains [BDN18]. Leveraging the homomorphic property of the underlying KZG polynomial commitment scheme, these key and signature aggregation techniques are also applicable to $\mathtt{DSKE}_{\mathsf{kzg}}$, offering a high level of space efficiency.

Reduce the Size of the Private Key. The private key in $\mathtt{DSKE}_{\mathsf{poly}}$, i.e., the polynomial $f(X)$, contains d elements of $\mathbb{F}$, i.e., the coefficients $f_0, \ldots, f_{d-1}$. A possible optimization is to employ a pseudorandom function (PRF) $F : \{0,1\}^\lambda \times \mathbb{Z} \to \mathbb{F}$ and pick a $k \in \{0,1\}^\lambda$ as the private key. The coefficients are derived as $f_i = F(k, i)$, for $i \in 0, \ldots, d-1$. In the security proof the challenger now chooses k uniformly at random and uses the PRF to

Table 3: Comparison of signature schemes with deniability.

Scheme	Without Requiring Future Actions (e.g., Publishing Keys)	Without Requiring External Services (e.g., Random Beacon)	Without Requiring VDF
KeyForge [SPG21]	✗	✓	✓
TimeForge [SPG21]	✓	✗	✓
Short-lived signature [ABC22]	✓	✗	✗
GroupForge$_{\mathsf{hash}}$ (based on $\mathtt{DSKE}_{\mathsf{hash}}$)	✓	✓	✓
GroupForge$_{\mathsf{poly}}$ (based on $\mathtt{DSKE}_{\mathsf{poly}}$)	✓	✓	✓

derive $f(X)$. From the security definition of PRF, $f(X)$ still looks random to the simulator $\mathcal{B}$. Observe, however, that $\mathsf{Extract}()$ would still return all d coefficients. Similarly, the KeyGen algorithm in GroupForge generates 2^h private keys for the underlying Σ_{DSKE} scheme, i.e., $\{sk_i\}_{i\in[2^h]}$, which are then stored in the private key SK. Using again a PRF $F' : \{0,1\}^\lambda \times \mathbb{Z} \to \mathbb{Z}$ and a private key $k' \in \{0,1\}^\lambda$ we can generate them as $sk_i = F'(k', i)$, for $i \in 1, \ldots, 2^h$, and store only k' in SK. As long as 2^h is polynomial in λ the outputs of F' are pseudorandom and security holds. We leave the formalization of these optimizations as future work.

Further Applications. DSKE can facilitate more applications where the need for authenticity is momentary and signers desire long-term deniability. Such examples are electronic voting and monetary donations. It has been argued that, even if future adversaries cannot go back to change the integrity or the result of a finished election, the privacy requirements of voters in the face of retaliation are everlasting [GCG+19]. Similarly, leaked donation records have reportedly caused individuals to be targeted and threatened [Ott22], and transaction deniability might be desirable in such scenarios as well.

9 Related Work

DAPS. Designing digital signature scheme with a key-extraction property has already been explored in the context of double authentication preventing signatures (DAPS). DAPS are genuinely designed with the purpose of double or multiple authentication prevention [PS14, RKS, BPS17, DRS18], and they particularly aim at messages of special form, namely $m = (a, p)$, where a is an address and p is a payload. In case a signer signs two (or more) messages with the same address but different payloads, its private key is leaked. However, DSKE essentially provides key extractability as an inherent feature without making assumptions about the type of message, increasing its applicability. In particular, the key extraction in our polynomial commitment-based DSKE (i.e., $\mathtt{DSKE}_{\mathsf{poly}}$) directly comes from the polynomial interpolation theorem. Note that one major downside of many DAPS schemes is their limited address space, i.e., exponentially large address space is not supported. Moreover, a notable difference between DSKE and DAPS is that the former has a sole design similar to the typical signature scheme in the literature like BLS [BLS01], while the latter has a hybrid design, containing different components. For instance, DAPS of [BPS17] builds on trapdoor identification schemes and that of [DRS18] involves encryption scheme and secret sharing. This, in turn, results in DAPS having considerably larger key pairs and also signature sizes compared to the normal signatures [DRS18].

GroupForge. Specter, Park, and Green formally defined the notion of forward-forgeable signature (FFS) [SPG21] and showed how FFS can be used to achieve deniability in the email protocol. The main idea of their scheme is to make the signatures become forgeable after a fixed delay. They present two concrete constructions: KeyForge and TimeForge. In the KeyForge construction, the email server needs to periodically publish expired keys to claim deniability over sent emails, and in the TimeForge construction, the signers need to rely on a trusted publicly verifiable time-keeping (PVTK) service to provide them with a verifiable clock time proof. The forgeability comes from the possibility of obtaining a valid proof by querying the PVTK service after a fixed delay.

Arun, Bonneau, and Clark proposed a similar notion to FFS called short-lived signatures [ABC22]. The main idea of their work is to leverage a disjunctive statement to achieve deniability, building up on the previous efforts in the literature, e.g., designated verifier proofs [JSI96], proofs-of-work-or-knowledge (PoWorKs) [BKZZ16], ring signatures [RST01], and one-out-of-many proofs [GK15]. In particular, the construction in Arun *et al.*'s work [ABC22] deploys verifiable delay functions [BBBF18, Wes19] as its main building block together with a (trusted) randomness beacon. They use a statement of the form: *I know the witness, x (e.g., private key), or someone solved a VDF on a beacon value derived from the trusted beacon before a time, t.* Hence, their construction offers deniability to the prover because anyone can produce a valid proof by evaluating the VDF through sequential computation. Moreover, a recent analysis report [LMPR23] of the VDF implementation shows that VDFs in practice, such as Minroot [KMT22], the VDF for the Ethereum blockchain, might be prone to various potential attacks, which have yet not been explored thoroughly. This work offers a simpler approach without requiring costly VDF evaluations and a trusted random beacon. We provide a comparison of our work and state-of-the-art signature schemes with deniability in Table 3.

Rate-Limiting Nullifier. A Rate-Limiting Nullifier (RLN) [Pri24] is a mechanism designed to restrict users in the number of actions within a system, thereby facilitating an effective spam prevention mechanism. RLN can be used in anonymous voting schemes [Bla21b], privacy-preserving P2P networks [Res21], and decentralized blockchain applications [Bla21a]. RLN has been extensively discussed by the blockchain community and implemented in practice [Lim23]. However, to our knowledge, no academic initiative has been undertaken to formally propose and establish secure RLN constructions as we do in this work with DSKE. In addition, it is feasible to extend DSKE-based RLN to an anonymous setting by employing non-interactive zero-knowledge proofs.

10 Conclusion

This paper introduces DSKE, a novel signature scheme featuring key extraction capabilities. We present concrete DSKE constructions based on the hash-based signature schemes and polynomial commitment schemes. We provide formal proof of security for our constructions, demonstrating that signers can consistently achieve deniability by presenting a set of signatures utilized to regenerate (old) private keys. Furthermore, we illustrate how DSKE can serve as the foundation for constructing both GroupForge signatures and RLN schemes. We posit that DSKE holds promise for application in other protocols that require short-term authenticity.

References

[ABC22] Arasu Arun, Joseph Bonneau, and Jeremy Clark. Short-lived zero-knowledge proofs and signatures. In *ASIACRYPT 2022*. Springer, 2022.

[ACD$^+$07] E Allman, Jon Callas, M Delany, Miles Libbey, J Fenton, and M Thomas. Domainkeys identified mail (dkim) signatures. Technical report, RFC 4871, May, 2007.

[AJM$^+$23] Ittai Abraham, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, and Gilad Stern. Bingo: Adaptivity and asynchrony in verifiable secret sharing and distributed key generation. In *Advances in Cryptology – CRYPTO*, 2023.

[BB21] Blagoj and WhiteHat Barry. Decentralised cloudflare - using rln and rich user identities, 2021. Available at: `https://ethresear.ch/t/decentralised-cloudflare-using-rln-and-rich-user-identities/10774`.

[BBB$^+$18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE symposium on security and privacy (SP)*, pages 315–334. IEEE, 2018.

[BBBF18] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In *Annual international cryptology conference*, 2018.

[BCG$^+$23] Gabrielle Beck, Arka Rai Choudhuri, Matthew Green, Abhishek Jain, and Pratyush Ranjan Tiwari. Time-deniable signatures. *Proceedings on Privacy Enhancing Technologies*, 2023.

[BDE$^+$11] Johannes Buchmann, Erik Dahmen, Sarah Ereth, Andreas Hülsing, and Markus Rückert. On the security of the winternitz one-time signature scheme. In *AFRICACRYPT 2011*. Springer, 2011.

[BDH11] Johannes Buchmann, Erik Dahmen, and Andreas Hülsing. XMSS - A Practical Forward Secure Signature Scheme Based on Minimal Security Assumptions. In Bo-Yin Yang, editor, *Post-Quantum Cryptography*, pages 117–129, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[BDN18] Dan Boneh, Manu Drijvers, and Gregory Neven. Compact multi-signatures for smaller blockchains. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 435–464. Springer, 2018.

[BGB04] Nikita Borisov, Ian Goldberg, and Eric Brewer. Off-the-record communication, or, why not to use pgp. In *Proceedings of the 2004 ACM workshop on Privacy in the electronic society*, pages 77–84, 2004.

[BKZZ16] Foteini Baldimtsi, Aggelos Kiayias, Thomas Zacharias, and Bingsheng Zhang. Indistinguishable proofs of work or knowledge. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 902–933. Springer, 2016.

[Bla21a] Blagoj. Private message sharing for eth2 validators, 2021. Available at: `https://ethresear.ch/t/private-message-sharing-for-eth2-validators/10664`.

[Bla21b] Blagoj. Rate limiting nullifier: A spam-protection mechanism for anonymous environments, 2021. Available at: `https://medium.com/privacy-scaling-explorations/rate-limiting-nullifier-a-spam-protection-mechanism-for-anonymous-environments-bbe4006a57d`.

[BLS01] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In *International conference on the theory* *and application of cryptology and information security*, pages 514–532. Springer, 2001.

[Bly05] Stephen E Blythe. Digital signature law of the united nations, european union, united kingdom and united states: Promotion of growth in e-commerce with enhanced security. *Richmond Journal of Law & Technology*, 11(2):6, 2005.

[BM99] Mihir Bellare and Sara K Miner. A forward-secure digital signature scheme. In *Annual international cryptology conference*, pages 431–448. Springer, 1999.

[BPS17] Mihir Bellare, Bertram Poettering, and Douglas Stebila. Deterring certificate subversion: efficient double-authentication-preventing signatures. In *Public-Key Cryptography–PKC 2017*, 2017.

[BS20] Dan Boneh and Victor Shoup. A graduate course in applied cryptography. *Draft 0.5*, 2020.

[DRS18] David Derler, Sebastian Ramacher, and Daniel Slamanig. Short double-and n-times-authentication-preventing signatures from ecdsa and more. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 273–287. IEEE, 2018.

[GBH17] Leon Groot Bruinderink and Andreas Hülsing. "oops, i did it again"–security of one-time signatures under two-message attacks. In *International Conference on Selected Areas in Cryptography*, pages 299–322. Springer, 2017.

[GCG$^+$19] Huangyi Ge, Sze Yiu Chau, Victor E Gonsalves, Huian Li, Tianhao Wang, Xukai Zou, and Ninghui Li. Koinonia: verifiable e-voting with long-term privacy. In *Proceedings of the 35th Annual Computer Security Applications Conference*, pages 270–285, 2019.

[GK15] Jens Groth and Markulf Kohlweiss. One-out-of-many proofs: Or how to leak a secret and spend a coin. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 253–280. Springer, 2015.

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald L Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on computing*, 17(2):281–308, 1988.

[Gre20] Matthew Green. Ok Google: please publish your DKIM secret keys, 2020. `https://blog.cryptographyengineering.com/2`

020/11/16/ok-google-please-publish-your-dkim-secret-keys/.

[JSI96] Markus Jakobsson, Kazue Sako, and Russell Impagliazzo. Designated verifier proofs and their applications. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 143–154. Springer, 1996.

[Kar19] Nikolaos Karanikolas. Digital signature legality in different jurisdictions: Legally binding issues. 2019.

[KMT22] Dmitry Khovratovich, Mary Maller, and Pratyush Ranjan Tiwari. Minroot: Candidate sequential function for ethereum vdf. *Cryptology ePrint Archive*, 2022.

[KZG10] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security*, volume 6477, pages 177–194. Springer, 2010.

[Lam79] Leslie Lamport. Constructing digital signatures from a one way function. 1979.

[Lim23] Wanseob Lim. Rln on kzg polynomial commitment scheme, 2023. Available at: https://zkresear.ch/t/rln-on-kzg-polynomial-commitment-scheme-cross-posted/114.

[LMPR23] Gaetan Leurent, Bart Mennink, Krzysztof Pietrzak, and Vincent Rijmen. Analysis of minroot: Public report, 2023. Available at: https://crypto.ethereum.org/events/minrootanalysis2023.pdf.

[Mas16] Stephen Mason. *Electronic signatures in law.* University of London Press, 2016.

[Mer89] Ralph C Merkle. A certified digital signature. In *Conference on the Theory and Application of Cryptology*, pages 218–238. Springer, 1989.

[NRBB22] Valeria Nikolaenko, Sam Ragsdale, Joseph Bonneau, and Dan Boneh. Powers-of-tau to the people: Decentralizing setup ceremonies. *Cryptology ePrint Archive*, 2022.

[Ott22] Ottawa Citizen. Threats close stella luna gelato café after owner's name appears in givesendgo data leak, 2022. https://ottawacitizen.com/news/local-news/threats-close-stella-lun a-gelato-cafe-after-owners-name-appears-in-givesendgo-data-leak.

[Pri24] Privacy and Scaling Explorations team, Ethereum Foundation. Rate-limiting nullifier, 2024. Available at: https://rate-limiting-nullifier.github.io/rln-docs/rln.html.

[PS14] Bertram Poettering and Douglas Stebila. Double-authentication-preventing signatures. In *Computer Security-ESORICS 2014: 19th European Symposium on Research in Computer Security, Wroclaw, Poland, September 7-11, 2014. Proceedings, Part I 19*, pages 436–453. Springer, 2014.

[Res21] Vac Research. Privacy-preserving p2p economic spam protection in waku v2, 2021. Available at: https://vac.dev/rlog/rln-relay/.

[RKS] Tim Ruffing, Aniket Kate, and Dominique Schröder. Liar, liar, coins on fire! penalizing equivocation by loss of bitcoins. In *CCS 2015*.

[RS04] Phillip Rogaway and Thomas Shrimpton. Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In *International workshop on fast software encryption*, pages 371–388. Springer, 2004.

[RST01] Ronald L Rivest, Adi Shamir, and Yael Tauman. How to leak a secret. In *International conference on the theory and application of cryptology and information security*, pages 552–565. Springer, 2001.

[SPG21] Michael A Specter, Sunoo Park, and Matthew Green. KeyForge:non-attributable email from Forward-Forgeable Signatures. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 1755–1773, 2021.

[Wes19] Benjamin Wesolowski. Efficient verifiable delay functions. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2019.

A Winternitz OTS-Based Construction

In the following, we demonstrate how hash-based Winternitz can also be a good candidate for DSKE construction.

Domination Free Function. A domination free function $P : \mathcal{M} \to I_w^\lambda$ determines the existential unforgeability and extractable set of our Winternitz Hash-based $\mathsf{DSKE_{wots}}$. We use the definition of the domination free function construction from [BS20].

Given a message $m \in \mathcal{M}$ and a hash function $H : \mathcal{M} \to \{0,1\}^\nu$, we compute $H(m)$ and convert it to an integer in $[0, 2^\nu)$. Given the public parameter w, let λ_0 be the smallest number satisfying $2^\nu \leq (w+1)^{\lambda_0}$, set $\lambda_1 = \log_{w+1}(w \cdot \lambda_0) + 1$ and $\lambda = \lambda_0 + \lambda_1$. Given an input message m, the function $P(m)$ works as follows:

- Compute $H(m)$, convert $H(m)$ to a λ_0-digit number in base $(w+1)$: $(s_1, \cdots, s_{\lambda_0})$.
- Compute the checksum $c = w \cdot \lambda_0 - (s_1 + \cdots + s_{\lambda_0})$.
- Convert c to a λ_1-digit number in base $(w+1)$: $(c_1, \cdots, c_{\lambda_1})$.
- Output $(s_1, \cdots, s_{\lambda_0}, c_1, \cdots, c_{\lambda_1})$.

In this case, the function P is domination free [BS20] and can map the message m to a vector $P(m) = (s_1, \cdots, s_{\lambda_0}, c_1, \cdots, c_{\lambda_1}) \in I_w^\lambda$.

Winternitz-Based DSKE. The Winternitz signature scheme is another hash-based signature scheme, that allows a trade-off between computation and the size of the signature. Winternitz-based DSKE, $\mathsf{DSKE_{wots}}$, consists of the following algorithms:

- $\mathsf{Setup}(1^\lambda)$: On input the security parameter λ, the algorithm outputs the public parameter, par, which contains: *(1)* a hash function, $f : \{0,1\}^\lambda \to \{0,1\}^\lambda$, chosen from a family of preimage-resistant hash functions, *(2)* a hash function, $H : \mathcal{M} \to \{0,1\}^\nu$, chosen from a family of collision-resistant hash function *(3)* a parameter integer w, and *(4)* a domination free function P parameterized by g and w, which maps a message m to a vector $\vec{s}$ of length λ, and each component of s is a number in $\{0, \cdots, w\}$, namely $P : \mathcal{M} \to I_w^\lambda$, where $I_w^\lambda = (\{0, \ldots, w\})^\lambda$. The public parameters are implicitly input to all subsequent algorithms.

- $\mathsf{KeyGen}(par)$: For each $i \in [\lambda]$, sample $sk_i \overset{\$}{\leftarrow} \{0,1\}^\lambda$; compute $pk_i = f^{(w)}(sk_i)$; and output the private key, $SK = (sk_i)_{i \in [\lambda]}$, and the public key, $PK = (pk_i)_{i \in [\lambda]}$.

- $\mathsf{Sign}(SK, m)$: Parse $SK = (sk_i)_{i \in [\lambda]}$; compute $\vec{s} = P(m) = (s_1, \cdots, s_\lambda) \in I_w^\lambda$; and output $\sigma = (f^{(s_1)}(sk_1), \cdots, f^{(s_\lambda)}(sk_\lambda))$.

- $\mathsf{Verify}(PK, m, \sigma)$: Parse $PK = (pk_i)_{i \in [\lambda]}$, $\sigma = (\sigma_i)_{i \in [\lambda]}$, and compute $\vec{s} = P(m) = (s_1, \cdots, s_\lambda) \in I_w^\lambda$. For all $i \in [\lambda]$, if $f^{(w - s_i)}(\sigma_i) \neq pk_i$, return 0. Otherwise, return 1.

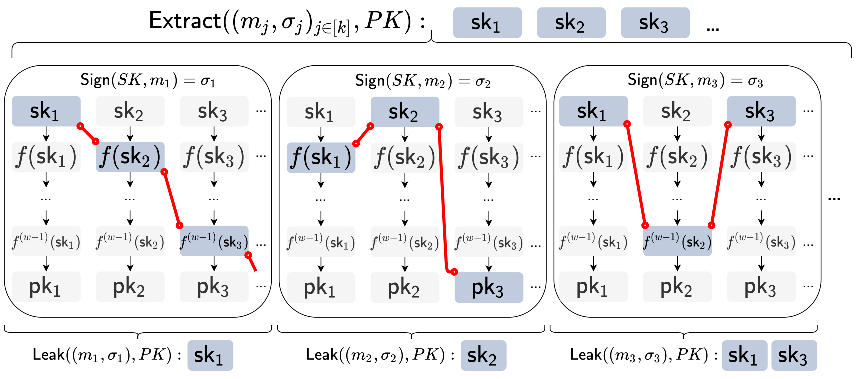

Extract$((m_j, \sigma_j)_{j\in[k]}, PK)$: sk$_1$ sk$_2$ sk$_3$ ...
Sign$(SK, m_1) = \sigma_1$
Sign$(SK, m_2) = \sigma_2$
Sign$(SK, m_3) = \sigma_3$
sk$_1$ sk$_2$ sk$_3$
f(sk$_1$) f(sk$_2$) f(sk$_3$)
$f^{(w-1)}$(sk$_1$) $f^{(w-1)}$(sk$_2$) $f^{(w-1)}$(sk$_3$)
pk$_1$ pk$_2$ pk$_3$
Leak$((m_1, \sigma_1), PK)$: sk$_1$
Leak$((m_2, \sigma_2), PK)$: sk$_2$
Leak$((m_3, \sigma_3), PK)$: sk$_1$ sk$_3$

Figure 4: Example of the algorithm Extract for DSKE$_{\mathsf{wtos}}$. The elements of a signature σ are linked via red lines. In this example, the original private key can be collectively reconstructed from $\{(m_i, \sigma_i)\}_{i\in[3]}$.

- Extract$(\{m_j, \sigma_j\}_{j\in[k]}, PK)$: For each message-signature pair (m_j, σ_j), compute $\vec{s}_j = P(m_j) = (s_{ji})_{i\in[\lambda]}$, and parse $\sigma_j = (\sigma_{ji})_{i\in[\lambda]}$. For each $j \in [k], i \in [\lambda]$, if $\exists s_{ji} = 0$, then let $sk_i = \sigma_{ji}$. Set $SK = (sk_i)_{i\in[\lambda]}$ and parse $PK = (pk_i)_{i\in[\lambda]}$. If $\forall i \in [\lambda]$, $pk_i = f^{(w-s_i)}(sk_i)$, then output SK. Otherwise, output $\perp$.

Leakage Function for Winternitz Signatures. As illustrated in Figure 4, given a Winternitz signature $\sigma = (\sigma_1, \cdots, \sigma_\lambda)$, the Leak() function of Winternitz signature will output all the signature elements $(\sigma_i)_i$ that are equal to the corresponding elements in the private key list, which satisfy $f^{(w)}(\sigma_i) = pk_i$.

- Leak$((m, \sigma), PK)$: Check if Verify$(PK, m, \sigma) = 1$; otherwise, output $\perp$. Compute $\vec{s} = P(m) = (s_i)_{i\in[\lambda]}$, and parse $\sigma = (\sigma_i)_{i\in[\lambda]}$. For each $i \in [\lambda]$, if $s_i = 0$, then let $S_i = \sigma_i$; otherwise, let $S_i = \perp$. Output $S = (S_i)_{i\in[\lambda]}$.

The intuition of the domination free function is to ensure that if a forger can obtain a valid signature for m' from one signature for m, domination property of P ensures that there exists one element s'_i in $P(m')$ that is smaller than $s_i \in P(m)$. Therefore, it implies the fact that such a forger can be used to invert the preimage-resistant hash function f.

Security Analysis. DSKE$_{\mathsf{wots}}$ satisfies unforgeability and extractability with trusted generation depending on the number of distinct message-signature pairs.

Theorem 7 (Existential Unforgeability)**.** *DSKE*$_{\mathsf{wots}}$ *is* existentially unforgeable under a 1-time adaptive chosen-message attack, *that is,*

$$\Pr[\mathsf{SignExp}^1_{\mathcal{A},DSKE_{\mathsf{wots}}}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$$

In the following analysis for the extractable set, we assume that the checksum computed by P is independent and uniformly random. This is not the case, because there is a dependency between the checksum elements and other elements in the vector s computed by P. However, as mentioned in [GBH17], without such an assumption, the analysis will become significantly more complex.

Lemma 2. *In a Winternitz-based DSKE scheme, if $H(\cdot)$ is modeled as a random oracle and the domination-free function $P(\cdot)$ outputs a uniformly random vector in I_w^λ, then the probability that an element of the private key will not be leaked after receiving k signatures is $\mathsf{p}_k^{leak} = \frac{w^k}{(w+1)^k}$.*

By combining Theorem 1 and Lemma 2, we can deduce Theorem 8.

Theorem 8 ($\mathtt{DSKE}_{\mathsf{wots}}$ Extractability with Trusted Generation)**.** *If $H(\cdot)$ is modeled as a random oracle and the domination free function $P(\cdot)$ outputs a uniformly random vector in I_w^λ, then $\mathtt{DSKE}_{\mathsf{wots}}$ has an (k, δ)-extractable set, where: $\delta \geq 1 - \frac{\lambda \cdot w^k}{(w+1)^k} - \mathsf{negl}(\lambda)$.*

B Missing Proofs

B.1 Proofs of Hashed-Based DSKE

In the following, we prove that $\mathtt{DSKE}_{\mathsf{lamp}}$ satisfies unforgeability and extractability with trusted generation depending on the number of distinct message-signature pairs.

Proof of Theorem 1.

Proof. Given a private key of a hash-based DSKE scheme, which is a list of λ elements, denoted as $sk = (sk_j)_{j \in [\lambda]}$. We denote Bad_0 to be the event that at least one element of sk is not included in $\bigcup_{i=1}^k \mathsf{Leak}((m_i, \sigma_i), pk)$. We have:

$$\begin{aligned}\Pr[\mathsf{Bad}_0] &= \Pr\left[\vee_{j=1}^{\lambda}\left(sk_j \notin \bigcup_{i=1}^{k} \mathsf{Leak}((m_i, \sigma_i), pk)\right)\right] \\ &\leq \sum_{j=1}^{\lambda} \Pr\left[sk_j \notin \bigcup_{i=1}^{k} \mathsf{Leak}((m_i, \sigma_i), pk)\right] = \lambda \cdot \mathsf{p}_k^{leak}\end{aligned}$$

Moreover, we denote Bad_1 to be the event that there are two private key elements that map to the same public key element. However, since the underlying hash is modeled as a random oracle, this probability is negligible: $\Pr[\mathsf{Bad}_1] = \mathsf{negl}(\lambda)$.

Therefore, the probability that the extraction algorithm $\mathsf{Extract}(\cdot)$ outputs a private key sk' such that $pk = \mathsf{extractPK}(sk')$ is $\delta \geq 1 - \Pr[\mathsf{Bad}_0 \vee \mathsf{Bad}_1] \geq 1 - \lambda \cdot \mathsf{p}_k^{leak} - \mathsf{negl}(\lambda)$. □

Missing Proof of Theorem 2.

Proof. **The t-Repeated One-Way Problem.** We first define the attack game of t-repeated one-way problem. Let f be a one-way function over $(\mathcal{X}, \mathcal{Y})$. For a given positive integer t and a given adversary $\mathcal{A}_1$, the game runs as follows:

- The challenger samples $x_1, \ldots, x_t \xleftarrow{\$} \mathcal{X}$, computes $y_1 \leftarrow f(x_1), \ldots, y_t \leftarrow f(x_t)$, and sends $(y_1, \ldots, y_t)$ to the adversary $\mathcal{A}_1$.
- The adversary $\mathcal{A}_1$ makes a sequence of reveal queries by sending indexes from $(1, \ldots, t)$ to the challenger. Upon receiving an index j, the challenger returns x_j to $\mathcal{A}_1$.
- $\mathcal{A}_1$ outputs (j^*, x), where $j^* \in (1, \ldots, t)$ and $x \in \mathcal{X}$.

We say that the adversary $\mathcal{A}_1$ wins the game if index j^* is not among $\mathcal{A}_1$'s reveal queries, and $f(x) = y_{j^*}$. We denote $\mathsf{rOWadv}[\mathcal{A}_1, f, t]$ as the probability that $\mathcal{A}_1$ wins the game.

According to Lemma 13.5 in [BS20], for every t-repeated one-way problem adversary $\mathcal{A}_1$, there exists an adversary $\mathcal{A}_0$ that can leverage $\mathcal{A}_1$ as a subroutine to break the preimage resistance of the function f. Moreover, let $\mathsf{OWadv}[\mathcal{A}_0, f]$ be the probability that $\mathcal{A}_0$ breaks the preimage resistance, we have $\mathsf{rOWadv}[\mathcal{A}, f, t] \leq t \cdot \mathsf{OWadv}[\mathcal{A}_0, f]$.

We now consider the adversary $\mathcal{A}$ that wins the **1-time signature experiment** $\mathsf{SignExp}^1_{\mathcal{A},\Sigma}(\lambda)$, and show that $\mathcal{A}$ can be used to solve the repeated one-way problem for f.

We construct an adversary $\mathcal{B}$ that uses $\mathcal{A}$ to win the repeated one-way game:

- The repeated one-way challenger $\mathcal{C}$ gives $\mathcal{B}$ a list of λ elements $y_1, \ldots, y_\lambda \in \{0,1\}^\lambda$. $\mathcal{B}$ aims to invert one of the elements.
- $\mathcal{B}$ sends $(y_1, \ldots, y_\lambda)$ as the public key pk to $\mathcal{A}$. Note that pk is indistinguishable from a public key generated by $\mathsf{KeyGen}()$.
- Upon receiving the signing request of a message m from $\mathcal{A}$, $\mathcal{B}$ requests from $\mathcal{C}$ the preimages of all i that $i \in H(m)$, and sends these preimages as the signature to $\mathcal{A}$.
- Eventually, $\mathcal{A}$ outputs a forgery σ^* for a message m^*, which is not already requested by $\mathcal{A}$, i.e., $m^* \neq m$. Let bad_1 be the event that $H(m^*) \neq H(m)$. Therefore, $\Pr[\mathsf{bad}_1]$ is negligible due to the collision-resistance of H.In this event, there exists some $j \in (1, \ldots, \lambda)$ which is not requested by $\mathcal{B}$, and if σ^* is a valid signature on m^* then σ^* contains a preimage x_j of y_j. $\mathcal{B}$ then outputs (j, x_j) as its solution to the repeated one-way problem.

Therefore, we have

$$\begin{aligned}&\Pr[\mathsf{SignExp}^{1}_{\mathcal{A},\mathsf{DSKE_{lamp}}}(\lambda)=1]\leq(1-\Pr[\mathsf{bad}_1])\cdot\mathsf{rOWadv}[\mathcal{A},f,t]+\Pr[\mathsf{bad}_1]\\&\leq\lambda\cdot\mathsf{OWadv}[\mathcal{A}_0,f]+\mathsf{negl}(\lambda)=\mathsf{negl}(\lambda)\end{aligned}$$

□

B.1.1 Missing Proof of Lemma 1.

Proof. We define $d_i = H(m_i)$. All d_i for $i \in \{1,\dots,k\}$ form a $k\times\lambda$ matrix:

$$\begin{pmatrix}H(m_1)\\ \vdots\\ H(m_k)\end{pmatrix}=\begin{pmatrix}d_1\\ \vdots\\ d_k\end{pmatrix}=\begin{pmatrix}d_{1,1}&\dots&d_{1,\lambda}\\ \vdots&\ddots&\vdots\\ d_{k,1}&\dots&d_{k,\lambda}\end{pmatrix}$$

where $d_{i,j}\in\{0,1\}$.

For $j\in[\lambda], b\in\{0,1\}$, to extract the value of $sk_j[b]$, there should be at least one d_{ij} that satisfies $d_{ij}=b$, i.e., for $j\in[\lambda]$, d_{ij} should not all be $1-b$. Therefore, to be able to extract an element sk_j, we do not want any columns in the matrix that contain all 0s or all 1s.

Since $H(\cdot)$ is modeled as a random oracle, p^{leak}_k is the probability that one single column consists of all 0s or all 1s:

$$\mathsf{p}^{leak}_k=\Pr\left[x\xleftarrow{\$}\{0,1\}^k : x=0^k\vee x=1^k\right]=\frac{1}{2^{k-1}}$$

□

B.2 Missing Proofs of Polynomial Commitment Schemes-Based DSKE

Missing Proof of Theorem 4.

Proof. Let $\mathcal{A}$ be a PPT adversary breaking the signature scheme $\mathtt{DSKE}_{\mathsf{poly}}$. We construct two PPT algorithms $\mathcal{B}_0$ and $\mathcal{B}_1$ that run $\mathcal{A}$ as a subroutine.

Algorithm $\mathcal{B}_0$ attacks the hiding property of the polynomial commitment scheme Π. From the challenger, $\mathcal{B}_0$ receives C_f, d, ck, vk, and up to d openings $(x_j, f(x_j), \pi_j)$, for $x_j\in\mathbb{F}$ and $j\in[d]$, and for $f(X)$ not known to $\mathcal{B}_0$. It breaks the hiding property if it outputs (x^*, y^*) for an unqueried x^* such that $y^*=f(x^*)$. It is important to note that C_f is of degree-d.

Algorithm $\mathcal{B}_1$ attacks the evaluation binding property of the polynomial commitment scheme Π. For d, ck, vk from the challenger, $\mathcal{B}_1$ breaks the evaluation binding property if it outputs $(C_f, x^*, (y,\pi), (y^*,\pi^*))$ such that $\mathsf{Check}(vk, C_f, x^*, (y,\pi))=1$, $\mathsf{Check}(vk, C_f, x^*, (y^*,\pi^*))=1$, and $y\neq y^*$. Breaking the evaluation binding property allows knowing $f(X)$ for $C_f=\mathsf{Com}(ck, f(X))$. It leads to a slight difference between $\mathcal{B}_0$ and $\mathcal{B}_1$

in the opening generation, where $\mathcal{B}_1$ generates the opening itself rather than relaying the opening from an existing challenger.

$\mathcal{B}$ takes $(d, (ck, vk)$ as the basic input. $\mathcal{B}_0$ additionally takes C_f and $(x_j, f(x_j), \pi_j)$ for $j \in [d]$. $\mathcal{B}_{b\in\{0,1\}}$ works as follows:

- If $b = 1$, C_f is empty, so it samples $f(X) \overset{\$}{\leftarrow} \mathbb{F}^d(X)$ and runs $\Pi.\mathsf{Com}(ck, f(X)) \to C_f$. Initiate $\mathcal{A}$ with $pk = C_f$ for C_f and create an empty set S_{quer}.
- For hash query on m, $\mathcal{B}$ samples $y, r \overset{\$}{\leftarrow} \mathbb{F}$, programs $H(m, r)$ as y, and replies (y, r).
- Whenever $\mathcal{A}$ requests a signature on message m_i: If $b = 1$, $(x_j, f(x_j), \pi_j)$ is not generated, so it samples $x_j \overset{\$}{\leftarrow} \mathbb{F}$ and runs $\Pi.\mathsf{Open}(ck, C_f, x_j, f(X)) \to (y_i, \pi_i)$. $\mathcal{B}$ samples $r_i \overset{\$}{\leftarrow} \mathbb{F}$, programs $H(m_i, r_i)$ as x_i from the openings, adds x_i to S_{quer}, and replies $\sigma_i = (\pi_i, y_i, r_i)$.
- If $\mathcal{A}$ fails to output a valid forgery on an unqueried message, then abort. Otherwise $\mathcal{A}$ has output a message m^* and a forgery $\sigma^* = (\pi^*, y^*, r^*)$ on m^*. We assume w.l.o.g. $\mathcal{A}$ has made d signature queries (if not, $\mathcal{B}$ queries these values itself) and hence $\mathcal{B}$ has openings (π_i, y_i) for points x_i, with $i \in [d]$. Compute $x^* = h(m^*, r^*)$. If $x^* \in S_{\text{quer}}$, then set $\mathsf{bad}_1 \leftarrow 1$ and abort. Otherwise interpolate $f'(X) \in \mathbb{F}^d(X)$ from the $d+1$ points $\{(x_1, y_1), \ldots, (x_d, y_d), (x^*, y^*)\}$ and compute $C_{f'} = \Pi.\mathsf{Com}(ck, f'(X))$. If $C_{f'} \neq C_f$, $\mathcal{B}_1$ runs $\Pi.\mathsf{Open}(ck, C_f, x^*, f(X)) \to (y, \pi)$, and outputs $(C_f, x^*, (y, \pi), (y^*, \pi^*))$ to break the evaluation binding property. Else, $\mathcal{B}_0$ outputs (x^*, y^*) to break the hiding property.

The event bad implies that $\mathcal{A}$ breaks the collision resistance property of H, which is assumed secure, hence $\Pr[\mathsf{bad}] = \mathsf{negl}(\lambda)$. $\mathcal{B}$ has two routes, depending if the interpolated polynomial $f'(X)$ matches the commitment C_f or not. If it does not match, i.e., $C_{f'} \neq \Pi.\mathsf{Com}(ck, f'(X))$, it implies that (x^*, y^*) is not a point of $f(X)$, i.e., $f(x^*) \neq y^*$. Since $\mathcal{A}$ succeeded, point (x^*, y^*) and proof π^* satisfy $\Pi.\mathsf{Check}(vk, C_f, x^*, y^*, \pi^*) = 1$. $\mathcal{B}_1$ outputs $(C_f, x^*, (y, \pi), (y^*, \pi^*))$ to breaks the evaluation binding property.

If it matches, i.e., $C_{f'} = \Pi.\mathsf{Com}(ck, f'(X))$, it implies that (x^*, y^*) is a point of $f(X)$, i.e., $f(x^*) = y^*$. $\mathcal{B}_0$ outputs (x^*, y^*) to breaks the hiding property. Note that although it is possible $f(X) \neq f'(X)$ for $C_{f'} = C_f$, it is impossible when both $f(X)$ and $f'(X)$ are of degree-d.

□

B.3 Missing proof of GroupForge

GroupForge satisfies the *correctness*, *unforgeability*, and *forgeability on expiry* properties of an FFS scheme.

Correctness follows from the underlying Σ_{DSKE} scheme and from the correctness of the Merkle proofs: every signature $\sigma = (s, pk_i, \mathsf{leaf}_i, \mathsf{proof}_i)$ constructed with $\mathsf{Sign}()$ will be valid according to $\Sigma_{\mathsf{DSKE}}.\mathsf{Verify}()$, and proof_i will be a valid Merkle proof that pk_i is the public key that corresponds to leaf_i.

Unforgeability is also reduced to the unforgeability of Σ_{DSKE} through standard arguments on Merkle-based constructions [BDH11]. Assume an adversary $\mathcal{A}$ that is given PK and attacks GroupForge. For each time chunk $i \in [2^h]$, $\mathcal{A}$ is allowed to make up to k signing queries. The challenger of $\mathcal{A}$ delegates these queries to Σ_{DSKE}. Assume a successful forgery $\sigma^* = (s^*, pk^*, \mathsf{leaf}^*, \mathsf{proof}^*)$, for some leaf leaf^*. Since Σ_{DSKE} is unforgeable, pk^* must be a public key under which s^* is a valid signature. In this case, however, the hash of pk^* and the Merkle proof proof^* can be used by the challenger to break the collision resistance of $H()$.

Forgeability on expiry is proven in the following theorem.

Theorem 9. *GroupForge achieves correctness, unforgeability, and forgeability on expiry properties.*

Proof. Let $\mathcal{D}$ be an adversary that is given a signature σ_b, for $b \in \{0,1\}$, and, as described in the definition of FFS, aims to distinguish the value of b. Signature σ_0 is created with the private key sk_i of some time chunk i, i.e., $\sigma_0 \leftarrow \mathsf{Sign}(sk_i, m)$. Signature σ_1 is created using the expiry information η, which in turn is computed from the additional material $\mathcal{T}$, i.e., $\eta \leftarrow \mathsf{Expire}(\mathcal{T})$ and $\sigma_1 \leftarrow \mathsf{Forge}(\eta, m)$. Let us assume that $\mathcal{T}$ contains k valid message-signature pairs, i.e., $\mathcal{T} = \{(m_j, \sigma_j)\}_{j\in[k]}$, that all m_j are pairwise different, and that all signatures are created in time chunk i, i.e., $\sigma_j = (s_j, pk_i, \mathsf{leaf}_i, \mathsf{proof}_i)$, for each $j \in [k]$. In this case, since the underlying DSKE scheme Σ_{DSKE} has (k,δ)-extractable sets, $\mathsf{Expire}(\mathcal{T})$ returns $h \neq \perp$ with probability δ. For this $h = (\mathsf{leaf}_i, \mathsf{proof}_i, sk'_i)$ we know from the properties of Σ_{DSKE} that $sk_i = sk'_i$, and $\mathsf{Forge}()$ will compute σ_1 by calling $s = \Sigma_{\mathsf{DSKE}}.\mathsf{Sign}(sk_i, m)$, which is identical to how σ_0 is created. Hence, $\mathcal{D}$ can only guess the value of b with probability $1/2$.

Overall, assuming that the signer has created enough message-signature pairs $\mathcal{T}$ with the properties described above, with probability δ, $\mathcal{D}$ can only at random guess the value of b, while with probability $1-\delta$ (when $\mathsf{Expire}(\mathcal{T})$ returns $\perp$), the challenger of $\mathcal{D}$ cannot produce a valid σ_1 and $\mathcal{D}$ can correctly guess b. The overall success probability of $\mathcal{D}$ is $P = 1 - \delta/2$. Observe that for the polynomial commitment-based DSKE scheme $\delta = 1 - \mathsf{negl}(\lambda)$, hence $P = 1/2 + \mathsf{negl}(\lambda)$, while for the hash-based DSKE scheme $\delta \geq 1 - \lambda/(2^{k-1}) + \mathsf{negl}(\lambda)$, hence P asymptotically approaches $1/2$ as k grows. □

Setting the Deniable Group Size k and Delay Δ. GroupForge requires that no recipients of k signatures collaborate within a time chunk Δ. This requirement can be made plausible by adjusting two variables. First, the size of the deniable group k can be set to a value large enough, for example, $\lceil n/2 \rceil$, where n is the number of recipient email servers. This would be equivalent to the assumption that no more than half of the nodes may be Byzantine, which is typical in distributed protocols. Second, time delay Δ can be set to a low value. Similar to that of KeyForge, we can assume an upper bound $\hat{\Delta}$ in the time required for email delivery and then set $\Delta = \hat{\Delta}$. Hence, even if the signature recipients collaborate, the forged signature has a high probability of reaching the recipient email server in the next time chunk.

Moreover, the value of k and the choice of the underlying DSKE scheme affect the probability of the algorithm $\mathsf{Forge}()$ outputting a signature and not $\perp$. For $\mathtt{DSKE}_{\mathsf{poly}}$ we have $\delta = 1$ for all $k \geq d+1$, and the signer can choose the size of the group by selecting the degree of the polynomial accordingly. For $\mathtt{DSKE}_{\mathsf{lamp}}$ the probability δ changes with k.