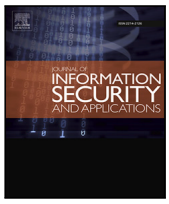
Journal of Information Security and Applications 91 (2025) 104058

Contents lists available at ScienceDirect

Journal of Information Security and Applications

journal homepage: www.elsevier.com/locate/jisa



Updatable Signature with public tokens

Haotian Yin [a], Jie Zhang [a,*], Wanxin Li [a,*], Yuji Dong [b], Eng Gee Lim [a], Dominik Wojtczak [c]

[a] School of Advanced Technology, Xi'an Jiaotong-Liverpool University, Suzhou, 215123, China
[b] School of Internet of Things, Xi'an Jiaotong-Liverpool University, Suzhou, 215400, China
[c] Department of Computer Science, University of Liverpool, Liverpool, L69 3BX, England, United Kingdom

ARTICLE INFO

Keywords:
Updatable signature
Web3
BLS signature

ABSTRACT

The Updatable Signature (US) allows valid signatures to be updated by an update token without accessing the newly generated signing key. Cini et al. (PKC'21) formally defined this signature and gave several constructions. However, their security model requires the secrecy of the update token, which is only applicable in some specific scenarios, such as software verification in the trusted App Store. In Web3, information is usually shared via a public blockchain, and decentralized private computation is expensive. In addition, one can use the same token to update both the signing key and signatures and all signatures can be updated with a single token. The adversarial signature generated by an adversary might also be updated. Therefore, this work explores the (im)possibility of constructing an Updatable Signature with public tokens (USpt), the tokens of which are signature-dependent. Specifically, we define the updatable signature with public tokens and present its security model. Then, we present a concrete USpt scheme based on the Boneh–Lynn–Shacham signature. This variant introduces a limitation for the signer who must maintain a dataset about its signed messages or hashes of them, which is applicable in our applications.

1. Introduction

A digital signature scheme is a cryptographic scheme that ensures information integrity, authentication, and non-repudiation. It allows the data sender to "sign" the information, and the receiver can verify the signature to confirm that the information is indeed from the claimed sender and has not been tampered with during transmission. In other words, a digital signature scheme can be designed to provide signature service through unforgeability under the circumstance that the adversary can acquire a list of message–signature pairs and is trying to forge a signature for some new message. Digital signature is an indispensable security tool in modern digital communications and file exchange and is widely used in software distribution, email, online transactions, and other scenarios that require high-security verification.

Post-compromise security is a relatively new security goal for digital signatures. This security property was formally defined by Cohn-Gordon, Cremers, and Garratt [1], who considered the security of Authenticated Key Exchange (AKE) protocols [2,3] after a party was fully compromised under some reasonable assumptions. Informally, in their adversary model with post-compromise security through the state, there must be at least one uncompromised session before the test session (the session key of which is picked as a challenge). In other words, post-compromise security considers the security after full compromise but adds some restrictions to the adversary afterwards. A similar security goal is forward security, which requires that subsequent compromises will not affect the security of any previous sessions. A common feature of forward and post-compromise security is that they consider the security of cryptographic schemes in terms of time sequence.

Updatable Signature (US) captures post-compromise security concerning signature schemes. Cini et al. [4] present the definition of US with secret tokens and its security model inspired by updatable encryption. Their model divides the operations of signers and verifiers into epochs, and in each epoch, different private keys are used for signing and public keys for verification. Suppose an adversary obtained some signer's signing key sk_e in epoch e. Then the adversary lost access to the signer's local state (this attack can be considered as a transient corruption [5] without controlling the signer); the signer is somewhat aware of this leakage and wants to update its local signing key to sk_{e+1}, which also advances the epoch to $e+1$. To keep the validity of previous signatures in epoch e, say a signature σ_e without loss of generality, the signer also generates a token Δ_{e+1} and *secretly* transfers it to the verifier who wants to update σ_e to a current-epoch-valid signature, σ_{e+1}.

* Corresponding author.
E-mail addresses: Jie.Zhang01@xjtlu.edu.cn (J. Zhang), Wanxin.Li@xjtlu.edu.cn (W. Li).

https://doi.org/10.1016/j.jisa.2025.104058

Available online 22 April 2025

Interestingly, unlike traditional signature schemes, this US definition introduces a trusted third-party *agent*, the receiver of the update token and responsible for updating the signatures instead of the verifiers.

1.1. Motivation

The US in [4] relies on a *trusted* third-party *agent*, which limits its application scenarios and might conflict with some system requirements, *e.g.*, a trustless blockchain application (see more specific applications in Section 5.1). The public blockchain virtual machine is a proxy/agent in this case. However, the update token can also be used to update the signing key. If it is public, the adversary can update the last epoch's signing key and continue forging valid signatures. Therefore, we seek to make token *public* and rely on a *semi-honest* agent in US schemes to break the above limitations.

This modification is significant for the application in Web3, where the signatures are usually managed by a public blockchain, and the computation on this agent is conducted in a decentralized manner. We detail discussed the application in Section 5.1.

1.2. Related and previous work

The Cini et al. framework [4] is well established for the US with *secret* tokens. They build their first construction based on Key-Homomorphic signature schemes [6], two features of which are the *homomorphism* of signing keys and the *perfect adaptability* of signatures (formally defined in Definition 4). They also designed US from Boneh–Lynn–Shacham (BLS) signature [7], equipping a property called Message Independence (MI): the update algorithm does not need to take the message as an input. This feature simplifies the update procedure since the verifier does not need to retrieve the original message. The security model rules out trivial attacks by a *leakage profile* [8], compromised by key-update inferences, token inferences, and signature-token inferences. Those records log the adversary's compromising queries, providing evidence to identify the challengable signatures and corresponding epochs.

Zhou and Liu [9] improved the US schemes by weakening the function of update tokens, which is similar to our goals, but their application scenario is the same as that of Cini et al. [4]. Specifically, their solution relies on a secure channel established by an AKE protocol [10] between the signer and a trusted agent (considered a server in their work). At the same time, the secure channel and extra protocols are significant issues we want to avoid, and deploying an AKE protocol also renders the final scheme interactive. Furthermore, their update process also utilized indistinguishability obfuscation [11,12], introducing additional computational overhead that is considerable relative to a signature scheme. They introduce a complete security analysis and proof [13] based on a more stringent CDH assumption recently, and the tokens are generated by the signer together with the proxy/server.

Bellare and Miner [14] devised the first forward-secure signature scheme. This scheme updates the signing key without modifying the verification key. An updated key-signed signature can be verified by the verification key with the corresponding epoch "timestamp". However, this kind of signature scheme does not care about the signature update; the signer must sign the message again even though this message was signed in the previous epoch. This work might not be technically relevant to the US because they have different goals, but the security notion is close but still different from the post-compromise security.

Beck et al. [15] abstract a new primitive, time-deniable signature. Each signature is associated with a timestamp, and the signer is able to disavow any signatures after a period of time. Their application is to facilitate email communication where ephemeral authentication is sufficient. Just like the forward-secure signature, a time-deniable signature does not care about the updatability and validity of "outdated" signatures. This notion serves short-term authentication, and its original intention is to prevent the non-repudiation of long-term authentication after data leakage or theft.

Chatzigiannis et al. [16] survey the most recent Web3 recovery mechanisms. Many crypto wallets are multi-party computed based on multi-signatures or threshold signatures, and the recovery mechanisms originate from the assumption of (at least) multi-keys' security. Those key rotation methods prevent future transactions using compromised or stolen keys; they cannot reverse transactions that have already occurred. However, the US rotates keys and signatures simultaneously; our signature-dependent updatability can also enable the signer deniability for illegal signatures. More specifically, the design idea of managerless group signature [17] is to spread the risk of key leakage, regardless of the data protected by the key. The nature of an updatable signature is to provide a convenient way to maintain the legitimacy of the signature, that is, how to safely cleanse the risk of leakage. KELP [18] provides a reactive way for key recovery based on smart contracts. They use staged and challenge-based contracts to reclaim the stolen or mis-transferred funds, which is not about key rotation but an engaging way of hindering attackers' behavior.

1.3. Updatable signature with public tokens

We informally introduce our definition of the Updatable Signature with public tokens (USpt) here.

Recall the original demand for key updating: (1) replacing the signing key; (2) in the meantime, existing signatures can be updated into valid new signatures using signature update tokens released by the signer to avoid resigning the messages using the updated signing key. The first requirement comes from the need for security, and the second requirement comes from the need for usability. Suppose there is some adversary $\mathcal{A}$ who has obtained a signing key sk_e, $\mathcal{A}$ can generate *valid* signatures, say σ'_e, in epoch e, but not the signature σ'_{e+1} valid in epoch $e+1$ since the key is replaced from requirement (1). Because our US scheme publishes update tokens publicly, $\mathcal{A}$ can always use this token to update its forged signature σ'_e to a *valid* signature σ'_{e+1} satisfying the requirement (2). Note that here, we refer *valid* to the signatures that can be verified by the signer's public key of the current epoch. Suppose we exclude this attack in our security model, *e.g.*, challenging a signature from updating a forged but expired signature. In that case, this definition is too weak for the practical scenarios: the adversary $\mathcal{A}$ is not even allowed to challenge the message chosen by itself.

In this case, we must prevent a public token from being exploited by adversaries. Our solution is under the assumption of a *signed-message database*: the signer maintains a database to record the messages (or hash of them) it signed for future updating the signatures that itself generated. Therefore, the signer can *distinguish* the signatures generated by itself from those forged by other parties even if its signing key is leaked, and the published tokens are only for signatures generated by itself. This assumption is not too strong for the application of US schemes: recall that adversaries are allowed to query a signing key of a signer who will not follow instructions from the adversaries. In other words, the adversaries can obtain *read* access to the signer but not *write* access. This assumption is also practical since the signer only stores the hash of signed messages, not all the messages, which is considerably longer than the hashes of it. This difference can be concluded as our signature-update tokens are *signature-dependent* and the signature-update tokens in [4] are *signature-independent*, similar to the notion in updatable encryption schemes [19]. We discussed the difficulties in constructing a public signature-independent token with a constant size in Section 5.

One may argue that this assumption might be too complex to deploy and is out of touch with reality. To understand this issue, we can imagine the signer as an artist who will sign each of his/her artwork and store the hash of the artwork and the signature as proof of originality. This signer will also store hashes of artworks by some means, such as the InterPlanetary File System (IPFS) [20] or a hard

disk. In this case, the signature has economic value and is valuable to consider the stateful signer. On the other hand, potential adversaries are also attracted by the economic value [21], increasing the necessity of this updatable notion. For any challenge epoch e^* and message m^*, we allow the adversary to query the signatures of any message other than m^* before epoch e^*. This is similar to basic Existential UnForgeability under adaptively Chosen Message Attacks (EUF-CMA) security, where an adversary cannot submit a signature already queried for as a challenge. However, in the USpt, a query to a signature in any epoch will cause it to be updated in all subsequent epochs because the update token is public. Therefore, we need to rule out this trivial attack.

1.4. Contribution

The contributions of this work can be concluded as follows:

1. define the model for updatable signature with public tokens and
2. construct a USpt based on BLS signature and prove its security.

The rest of the paper will be organized as follows. Section 2 will introduce some common terminology, define key homomorphic signatures, and discuss security concepts related to signatures. Section 3 will introduce updatable signatures and how we define security when the token is public. We construct a USpt in Section 4 and prove its security. Finally, Section 5 will discuss the applications in Web3 and conclude the future work.

2. Preliminaries

2.1. Notation

For $n \in \mathbb{N}$, let $[n] := \{1, \dots, n\}$, and let $\lambda \in \mathbb{N}$ be the security parameter. For a finite set $\mathcal{S}$, we denote by $s \leftarrow \mathcal{S}$ the process of sampling s uniformly from $\mathcal{S}$. Similarly, for an algorithm A, we denote by $y \leftarrow \mathsf{A}(x)$ be the process of running A on input x with access to uniformly random coins and assigning the result to y. To make the random coins r explicit, we write $\mathsf{A}(x; r)$. We use $\perp$ to indicate that an algorithm terminates with an error and A^{B} when A has access to call B, where B may return $\top$ as a distinguished special symbol. We say that an algorithm A runs in Probabilistic Polynomial Time (PPT) if the running time of A is polynomial in security parameter λ. For some signature schemes, we denote the sets of signing keys, public keys, key-update tokens, and signature-update tokens by $\mathcal{SK}, \mathcal{PK}, \mathcal{KT}$, and $\mathcal{ST}$.

Let F be a binary relationship. We denote by $\mathcal{X}$ as the domain of F, $\mathcal{Y}$ as the range of F. We say F is a *function*, if $\forall x \in \mathcal{X}$, $\exists$ unique $y \in \mathcal{Y}$ s.t. xFy; when we treat some elements x' of the domain as constants, we write the constants as the subscript $F_{x'}$. For function F, if xFy, we write it as $y = F(x)$. We say a function $F : \mathcal{X} \to \mathcal{Y}$, with domain X and range Y, is *bijective*, if $\forall y \in \mathcal{Y}$, $\exists$ unique $x \in \mathcal{X}$ s.t. $y = F(x)$. In this case, the *inverse function* of F is the function $F^{-1} : \mathcal{Y} \to \mathcal{X}$. Given two functions $g : \mathcal{X} \to \mathcal{Y}$ and $h : \mathcal{Y} \to \mathcal{Z}$, their *composition* is the function $h \circ g : \mathcal{X} \to \mathcal{Z}$ defined by $(h \circ g)(x) = h(g(x))$.

2.2. Key-homomorphic signature

Derler and Slamanig [6,22] defined the Key-Homomorphic (KH) signature scheme, which is the main foundation of US with secret tokens [4]. We recall relevant definitions here.

Definition 1 (*Signature Scheme*)**.** A signature scheme Σ is a triple $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ of PPT algorithms, which are defined as follows:

- $\mathsf{Gen}(\lambda)$: This algorithm takes a security parameter λ as input and outputs a secret (signing) key sk and a public (verification) key pk with associated message space $\mathcal{M}$.
- $\mathsf{Sig}(sk, m)$: This algorithm takes a secret key sk and a message $m \in \mathcal{M}$ as input and outputs a signature σ.
- $\mathsf{Ver}(pk, m, \sigma)$: This algorithm takes a public key pk, a message $m \in \mathcal{M}$ and a signature σ as input and outputs a verdict $b \in \{0, 1\}$.

Definition 2 (*Secret Key to Public Key Homomorphism*)**.** A signature scheme Σ provides a secret key to public key homomorphism if there exists an efficiently computable map $\mu : \mathbb{H} \to \mathbb{E}$ such that for all $sk, sk' \in \mathbb{H}$ it holds that $\mu(sk + sk') = \mu(sk) \cdot \mu(sk')$, and for all $(sk, pk) \leftarrow \mathsf{Gen}(\lambda)$, it holds that $pk = \mu(sk)$.

Definition 3 (*Key-Homomorphic Signatures*)**.** A signature scheme is called key-homomorphic signature if it provides a secret key to public key homomorphism and an additional PPT algorithm Adapt, defined as:

- $\mathsf{Adapt}(pk, m, \sigma, \delta)$: Given a public key pk, a message m, a signature σ, and a shift amount δ, this algorithm outputs a public key pk' and a signature σ',

such that for all $\delta \in \mathbb{H}$ and all $(pk, sk) \leftarrow \mathsf{Gen}(\lambda)$, all messages $m \in \mathcal{M}$ and all σ with $\mathsf{Verify}(pk, m, \sigma) = 1$ and $(pk', \sigma') \leftarrow \mathsf{Adapt}(pk, m, \sigma, \delta)$ it holds that

$$(\Pr[\mathsf{Ver}(pk', m, \sigma') = 1] = 1) \wedge (pk' = \mu(\delta) \cdot pk).$$

Definition 4 (*Perfect Adaption*)**.** A key-homomorphic signature scheme provides perfect adaption if for every $\lambda \in \mathbb{N}$, every message $m \in \mathcal{M}$, it holds that

$$[\sigma, (sk, pk), \mathsf{Adapt}(pk, m, \sigma, \delta)],$$

where $(sk, pk) \leftarrow \mathsf{Gen}(\lambda), \sigma \leftarrow \mathsf{Sig}(sk, m), \delta \leftarrow \mathbb{H}$, and

$$[\sigma, (sk, \mu(sk)), (\mu(sk) \cdot \mu(\delta), \mathsf{Sig}(sk + \delta, m))],$$

where $sk \leftarrow \mathbb{H}, \sigma \leftarrow \mathsf{Sig}(sk, m), \delta \leftarrow \mathbb{H}$, are identically distributed.

US from [4] is mainly built on the algorithm Adapt, and the perfect adaption ensures the unlinkable updates (Definition 10) and indistinguishable simulation in security proof.

The most basic security requirement for a signature scheme is universal unforgeability under No-Message Attacks (UUF-NMA).

Definition 5 (*UUF-NMA*)**.** A signature scheme Σ is UUF-NMA secure iff for all PPT adversary $\mathcal{A}$ such that

$$\Pr\left[(sk, pk) \leftarrow \mathsf{Gen}(\lambda), m^* \leftarrow \mathcal{M}, \sigma^* \leftarrow \mathcal{A}(pk, m^*, \lambda) : \mathsf{Ver}(pk, m^*, \sigma^*) = 1\right]$$

is negligible in λ.

We also require EUF-CMA security.

Definition 6 (*EUF-CMA*)**.** A signature scheme Σ is EUF-CMA secure iff for any valid PPT adversary $\mathcal{A}$ such that

$$\mathsf{Adv}^{\mathsf{euf\text{-}cma}}_{\Sigma,\mathcal{A}}(\lambda) := \Pr\left[\mathsf{Exp}^{\mathsf{euf\text{-}cma}}_{\Sigma,\mathcal{A}}(\lambda) = 1\right]$$

is negligible in security parameter λ where $\mathsf{Exp}^{\mathsf{euf\text{-}cma}}_{\Sigma,\mathcal{A}}(\lambda) = 1$ is defined in Fig. 1.

3. Updatable signature with public tokens

This section defines USpt and its security model.

3.1. Defining the updatable signature with public tokens

First, let us review the definition of US with secret tokens in [4].

Definition 7 (*Updatable Signature*)**.** An US scheme US is a tuple of the five PPT algorithms $(\mathsf{Setup}, \mathsf{Next}, \mathsf{Sig}, \mathsf{Update}, \mathsf{Ver})$:

Experiment $\mathsf{Exp}^{\mathsf{euf\text{-}cma}}_{\Sigma,\mathcal{A}}(\lambda)$:
$(pk, sk) \leftarrow \mathsf{Gen}(\lambda)$
$(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{sig}(sk,\cdot)}(pk, \lambda)$
if $\mathsf{sig}(sk, m^*)$ is not queried, return $\mathsf{Ver}(pk, m^*, \sigma^*)$; otherwise, return 0

Fig. 1. The EUF-CMA security experiment.

- $\mathsf{Setup}(\lambda, n)$: On input security parameter λ and the maximum number of epochs $n \in O(2^\lambda)$, the setup algorithm outputs a public and secret key pair (pk_1, sk_1).
- $\mathsf{Next}(pk_e, sk_e)$: On input a public key pk_e and signing key sk_e for epoch $e \in [n-1]$, the key-update algorithm outputs an updated public key pk_{e+1}, an updated signing key sk_{e+1}, and an update token Δ_{e+1}.
- $\mathsf{Sig}(sk_e, m)$: On input signing key sk_e for epoch $e \in [n]$ and a message $m \in \mathcal{M}$, the signing algorithm outputs a signature σ_e.
- $\mathsf{UpdateS}(\Delta_{e+1}, m, \sigma_e)$: On input an update token Δ_{e+1}, a message m, and a signature σ_e for epoch $e < n$, the update algorithm outputs an updated message–signature pair (m, σ_{e+1}) or $\perp$.
- $\mathsf{Ver}(pk_e, m, \sigma_e)$: On input public key pk_e, a message m, and a signature σ_e for epoch $e \in [n]$, the verification algorithm outputs a verdict $b \in \{0, 1\}$.

In the US, we have two kinds of signatures: the signature generated by Sig algorithm and the signatures generated by UpdateS algorithm. We refer to the former signatures (generated by Sig) as *fresh* signatures and the latter signatures (generated by UpdateS) as *updated* signatures.

Now, let us introduce the USpt. Intuitively, we want to update a signing key sk_e to an updated key sk_{e+1} first, then update signatures under outdated key σ_e to new signatures under the updated key σ_{e+1}. Since we are trying to update different subjects in the above-mentioned two steps, we define two kinds of tokens:

- δ: key-update token;
- Δ: signature-update token.

We model the signed message database as a set of messages denoted by M. The key-update token δ is generated by algorithm Next and used firstly as an element to compute the following signing key. Then, the algorithm Next takes δ and M as input to compute a signature-update token Δ. In our definition, the key-update token δ is only used as an *intermediate* for computing the next public-secret key pair. We also need to add a verification step in the update signature algorithm UpdateS to limit the signature-update token only to update signed messages in set M.

Definition 8 (*Updatable Signature with Public Tokens*)**.** An USpt scheme USpt is a tuple of the PPT algorithms (Setup, Next, Sig, UpdateS, Ver):

- $\mathsf{Setup}(\lambda, n)$: On input security parameter λ and the maximum number of epochs $n \in O(2^\lambda)$, the setup algorithm outputs a public and secret key pair (pk_1, sk_1) and an initially empty signed-message set M.
- $\mathsf{Next}(pk_e, sk_e, M)$: On input a public key pk_e, signing key sk_e for epoch $e \in [n-1]$, and a signed-message set M, the next-key algorithm computes an updated public key pk_{e+1}, an updated secret key sk_{e+1}, and a signature-update token Δ_{e+1}.
- $\mathsf{Sig}(sk_e, m)$: On input signing key sk_e for epoch $e \in [n]$ and a message $m \in \mathcal{M}$, the signing algorithm outputs a signature σ_e; update the set $M \leftarrow \{m\} \cup M$.
- $\mathsf{UpdateS}(\Delta_{e+1}, m, \sigma_e)$: On input a signature-update token Δ_{e+1}, a message m, and a signature σ_e for epoch $e < n$, the update algorithm outputs a message and updated message–signature pair (m, σ_{e+1}) if $m \in M$, or $\perp$ if $m \notin M$.
- $\mathsf{Ver}(pk_e, m, \sigma_e)$: On input public key pk_e, a message m, and a signature σ_e for epoch $e \in [n]$, the verification algorithm outputs a verdict $b \in \{0, 1\}$.

3.1.1. Correctness of USpt

For all $\lambda, n \in \mathbb{N}$, for all $(pk_1, sk_1) \leftarrow \mathsf{Setup}(\lambda, n)$, for all $e \in [n-1]$, for all $(pk_{e+1}, sk_{e+1}, \Delta_{e+1}) \leftarrow \mathsf{Next}(pk_e, sk_e, M)$, for all σ_e with $\mathsf{Ver}(pk_e, m, \sigma_e) = 1$, for all $(m, \sigma_{e+1}) \leftarrow \mathsf{UpdateS}(\Delta_{e+1}, m, \sigma_e)$, we have that $\Pr[\mathsf{Ver}(pk_{e'}, m, \sigma_{e'}) \neq 1] \leq \epsilon(\lambda)$ holds, for all $e' \in [n]$, where $\epsilon(\lambda) = \mathsf{negl}(\lambda)$, and we call it perfectly correct if $\epsilon(\lambda) = 0$.

3.2. Security model

3.2.1. Global state

In our security game/experiments, let $q \in \mathbb{N}$ be the number of signature queries and e be the identifier of the current epoch. Next, we define a global state $\mathbf{S} = (\mathcal{I}, \mathcal{K}, \mathcal{S})$:

- $\mathcal{I} = \{((pk_{e'}, sk_{e'}), \Delta_{e'})_{e' \in [e]}\}$: all keys and signature-update tokens;
- $\mathcal{K} = \{e' \in [e]\}$: all epochs where the adversary queried $\mathsf{corrupt}(e')$;
- $\mathcal{S} = \{(e', m, \sigma_{e'})_{e' \in [e]}\}$: all tuples where the adversary queried $\mathsf{sig}(m, e')$ in epoch e' or $\mathsf{updates}(\Delta_{e'}, m, \cdot)$ in epoch $e' - 1$.

The set $\mathcal{I}$ is initialized by $\{((pk_1, sk_1), \Delta_1)\}$, where $(pk_1, sk_1) \leftarrow \mathsf{Setup}(\lambda, n)$ and $\Delta_1 := \perp$, and other three sets $\mathcal{K}, \mathcal{T}$ and $\mathcal{S}$ are initialized as empty.

Set $\mathcal{I}$ is for recording the key pairs and key-update tokens; set $\mathcal{K}$ is for recording the epochs that the signing keys of which have been obtained by the adversary; set $\mathcal{S}$ is also used for recording the adversarial signature requests, *e.g.*, logging the signatures queried by the adversary and corresponding epochs and messages. Unlike the US security model [4], we do not need to record the intermediate key-update token δ since it is not helpful for future signing and key updates. In other words, δ is not used as an intermediate value and is not transmitted via any channels; also, our signature-update token is publicly queriable.

With access to the global state $\mathbf{S} = (\mathcal{I}, \mathcal{K}, \mathcal{S})$, for any epoch $e' \in [e]$ the adversary is also given the following oracles (sig, next, updates, corrupt, ver):

$\mathsf{sig}(m, e')$: On input message m and epoch e', compute signature $\sigma_{e'} \leftarrow \mathsf{Sig}(sk_{e'}, m)$, update $\mathcal{S} := \mathcal{S} \cup \{(e', m, \sigma_{e'})\}$, and return $\sigma_{e'}$.
next : Find $(pk_e, sk_e) \in \mathcal{I}$, compute $(pk_{e+1}, sk_{e+1}, \Delta_{e+1}) \leftarrow \mathsf{Next}(pk_e, sk_e)$, update $\mathcal{I} := \mathcal{I} \cup \{((pk_{e+1}, sk_{e+1}), \Delta_{e+1})\}$, return (pk_{e+1}, Δ_{e+1}) and set $e := e + 1$.
$\mathsf{updates}(\Delta_{e'+1}, m, \sigma_{e'})$: On input a token-signature pair $(\Delta_{e'+1}, \sigma_{e'})$, return $\perp$ if $\mathsf{ver}(m, \sigma_{e'}) \neq 1$; otherwise, compute $t \leftarrow \mathsf{UpdateS}(\Delta_{e'+1}, m, \sigma_{e'})$, return $\perp$ if $t = \perp$; otherwise, parse $t = (m, \sigma_{e'+1})$, update $\mathcal{S} := \mathcal{S} \cup \{(e'+1, m, \sigma_{e'+1})\}$ and return $(m, \sigma_{e'+1})$.
$\mathsf{corrupt}(e')$: On input an epoch $e' \in [e]$, return $\perp$ if $e' \geq e$; otherwise, return $sk_{e'}$ and update $\mathcal{K} := \mathcal{K} \cup \{e'\}$.
$\mathsf{ver}(m, \sigma_{e'})$: On input a message–signature pair $(m, \sigma_{e'})$, return $b := \mathsf{Ver}(pk_{e'}, m, \sigma_{e'})$.

Experiment $\mathsf{Exp}^{\text{uspt-euf-cma}}_{\mathsf{USpt},\mathcal{A}}$:
$(pk_1, sk_1) \leftarrow \mathsf{Setup}(\lambda, n)$
$\mathbf{S} = (\mathcal{I}, \mathcal{K}, \mathcal{S}),$ for $\mathcal{I} := \{((pk_1, sk_1), \perp)\}, \mathcal{K} := \mathcal{S} := \emptyset$
$(m^*, \sigma^*_{e^*}, e^*) \leftarrow \mathcal{A}^{\mathsf{sig},\mathsf{next},\mathsf{updates},\mathsf{corrupt},\mathsf{ver}}(\lambda)$
if for some $e' \leq e^*, (e', m^*, \cdot) \in \mathcal{S}$ or $e^* \in \mathcal{K}$, return 0; otherwise,
return $\mathsf{Ver}(pk_{e^*}, m^*, \delta_{e^*}) = 1$

Fig. 2. USpt-EUF-CMA security experiment.

3.2.2. Leakage profile

Considering the relevant signatures between epochs, different from [4], our signature-update token is public; therefore, any valid signature in any previous epoch can be updated to the current activated epoch if one logs all the signature-update tokens from the previous epoch to now. In this case, we only need to consider whether the adversary can forge a different signature–message pair when given all valid message–signature pairs obtained from sig. This requirement is similar to the basic EUF-CMA security of plain signature schemes.

However, we should also restrict the *unchallengeable* signatures when the signing key is leaked. Ideally, one signing key leakage should only affect the security in that epoch, which is why we need such an updatable property. Therefore, the global state $\mathcal{K}$ and $\mathcal{S}$ is sufficient for recording all adversarial operations considering public tokens. This limitation can be presented in the following security definition.

3.2.3. Existential unforgeability under chosen-message attacks (USpt-EUF-CMA)

Informally, the USpt-EUF-CMA captures the security notions that no PPT adversary can forge a valid signature, even given the compromising oracles. We say that a USpt scheme USpt is USpt-EUF-CMA-secure if no PPT adversary $\mathcal{A}$ succeeds in the USpt-EUF-CMA security experiment (Fig. 2) with non-negligible probability.

To keep the adversary valid, it cannot query sig on the challenge message m^* on any epoch before e^* since the signature-update tokens are public, any signature in an epoch earlier than e^* can be updated to e^* by any adversary. On the contrary, signatures in an epoch greater than e^* cannot be updated to e^*. In addition, the signing key has to be kept secret in epoch e^*.

Definition 9 (*USpt-EUF-CMA security*)**.** A USpt scheme USpt is USpt-EUF-CMA-secure iff for any PPT adversary $\mathcal{A}$ the advantage function

$$\mathsf{Adv}^{\text{uspt-euf-cma}}_{\mathsf{USpt},\mathcal{A}}(\lambda, n) := \Pr\left[\mathsf{Exp}^{\text{uspt-euf-cma}}_{\mathsf{USpt},\mathcal{A}}(\lambda, n) = 1\right],$$

is negligible in λ, where $\mathsf{Exp}^{\text{uspt-euf-cma}}_{\mathsf{USpt},\mathcal{A}}(\lambda, n)$ is defined in Fig . 2.

3.2.4. Unlinkable updates under chosen-message attacks (USpt-UU-CMA)

Informally, USpt-UU-CMA captures the security notions that no PPT adversary can distinguish fresh signatures from updated signatures, even given all signing keys, key/signature-update tokens, and signatures from past epochs. We say that an USpt scheme USpt is USpt-UU-CMA-secure if any PPT adversary $\mathcal{A}$ succeeds in the USpt-UU-CMA security experiment (Fig. 3) with negligible advantage.

Definition 10 (*USpt-UU-CMA security*)**.** A USpt scheme USpt is USpt-UU-CMA-secure iff for any valid PPT adversary $\mathcal{A}$ the advantage function

$$\mathsf{Adv}^{\text{uspt-uu-cma}}_{\mathsf{USpt},\mathcal{A}}(\lambda, n) := \left|\Pr\left[\mathsf{Exp}^{\text{uspt-uu-cma}}_{\mathsf{USpt},\mathcal{A}}(\lambda, n) = 1 - \frac{1}{2}\right]\right|,$$

is negligible in λ, where $\mathsf{Exp}^{\text{uspt-uu-cma}}_{\mathsf{USpt},\mathcal{A}}(\lambda, n)$ is defined in Fig. 3.

4. Construction of USpt from the BLS signature

This section introduces our solution for constructing a USpt from the BLS signature. Some impossibility results are shown in Appendix A.

4.1. USpt from BLS signatures

The USpt scheme BLS-USpt is shown in Fig. 4. The signed message database for the signer can be more efficiently constructed by replacing the input message m to the hash of it, $H(m)$, therefore saving the storage of the signer. This method of publishing signature-token will not affect the security of key-token since deriving key-token δ_{e+1} from $H(m)^{\delta_{e+1}}$ is solving an instance of discrete logarithm problem in group $\mathbb{G}_1$. In this case, the signer wanting to update the signing keys and keep the validity of previously generated signatures must record hashes of the messages signed. For every signature update, the signer needs to add an element $H(m)^{\delta_{e+1}}$ to the signature-update token Δ_{e+1}. We deem this difference between the existing US schemes [4] as a new scenario application. Indeed, considering a trusted agent to update every signature on behalf of the signer, the signature-independent token is much easier to use and reduces the bandwidth during the process. Signature-dependent tokens still enable efficient updates in a trustless or decentralized scenario.

This method is surprisingly related to the notion of signature aggregation [23,24]. In the scenario of signature aggregation, given n signatures $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n)$ for message $m = (m_1, m_2, \ldots, m_n)$ from n signers with public key $pk = (pk_1, pk_2, \ldots, pk_n)$, the verifier is able to verify those n signatures together in just one call to aggregate verification algorithm AVer:

$\mathsf{AVer}(\sigma, pk, m)$:

- Compute $\sigma' = \prod_{i=1}^{n} \delta_i$.
- Compute $h_i \leftarrow H(m_i)$ for all $1 \leq i \leq n$.
- Return $\prod_{i=1}^{n} e(h_i, pk_i) = e(\sigma', \tilde{g})$.

In the case of updatable signatures, suppose the signer wants to update a signature $\sigma_e = H(m)^{sk_e}$ held by a verifier; the signer sends to the verifier a signature-update token, which contains a corresponding update token $(H(m)^{\delta_{e+1}}, pk_{e+1})$. The verifier updates the σ_e to $\sigma_{e+1} = \sigma_e \cdot H(m)^{\delta_{e+1}}$. Note that $H(m)^{\delta_{e+1}}$ is actually a signature under signing key δ_{e+1}. Finally, the verification algorithm is the same as single signature verification: return $e(H(m), pk_{e+1}) = e(\sigma_{e+1}, \tilde{g})$. This verifies a message m by aggregating two different public keys $pk_{e+1} = pk_e \cdot \tilde{g}^{\delta_e}$.

A basic requirement of BLS aggregation is that messages in that aggregation should be distinct. Regarding updatable signatures, the verifier can verify signatures on the same message in different epochs. This is not a problem since we assume the update tokens are authenticated. We will show the details in the formal proof.

Experiment $\mathsf{Exp}^{\mathsf{uspt\text{-}uu\text{-}cma}}_{\mathsf{USpt},\mathcal{A}}$:

$(pk_1, sk_1) \leftarrow \mathsf{Setup}(\lambda, n)$

$\mathbf{S} = (\mathcal{I}, \mathcal{K}, \mathcal{S}),$ for $\mathcal{I} := \{((pk_1, sk_1), \perp)\}, \mathcal{K} := \mathcal{S} := \emptyset$

$(m^*, e', e^*) \leftarrow \mathcal{A}^{\mathsf{sig,next,updates,corrupt,ver}}(\lambda)$

$b \leftarrow \{0, 1\}$

$\sigma^0 \leftarrow \mathsf{Sig}(sk_e^*, m^*), \sigma^1 \leftarrow \mathsf{UpdateSCh}(m^*, e', e^*)^a$

$b^* \leftarrow \mathcal{A}^{\mathsf{sig,next,updates,corrupt,ver}}(\lambda, \sigma^b)$

if $(e', m^*, \cdot) \in \mathcal{S}, e' < e^*$, and $b^* = b$, return 1; otherwise, return 0

[a] We use a compact notation $\mathsf{UpdateSCh}(m^*, e', e^*)$ to denote the repeated application of $\mathsf{updates}$ starting from epoch e' to forward the signature $\sigma_{e'}$ to the final signature σ_{e^*} in epoch e^*, then we assign σ_{e^*} to σ^1

Fig. 3. USpt-UU-CMA security experiment.

$\mathsf{Setup}(\lambda, n)$:

- Choose three groups $\mathbb{G}_1 = \langle g \rangle, \mathbb{G}_2 = \langle \tilde{g} \rangle$, and $\mathbb{G}_T$ are three groups of prime order p with $\lambda = \log_2 p$; construct a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$; choose a hash function $H : \mathcal{M} \to \mathbb{G}_1$ uniformly at random from hash function family $\{H_k\}_k$; set an empty signed-message-hash set $\mathcal{H} := \emptyset$.
- Choose $sk \leftarrow \mathbb{Z}_p^*$, and set $pk := \tilde{g}^{sk}$.
- Set $(pk_1, sk_1) := (pk, sk)$.
- Return $(pk_1, sk_1, \mathcal{H})$.

$\mathsf{Next}(pk_e, sk_e, \mathcal{H})$:

- Choose $\delta_{e+1} \leftarrow \mathbb{Z}_p^*$.
- $\mathsf{UpdateK}(\delta_{e+1}, sk_e)$: Compute $sk_{e+1} \leftarrow sk_e + \delta_{e+1} \mod p$.
- Compute $pk_{e+1} \leftarrow \tilde{g}^{sk_{e+1}}$.
- Compute $h_i^{\delta_{e+1}}$ for all $h_i \in \mathcal{H}$.
- Set signature-update token $\Delta_{e+1} := (h_i, h_i^{\delta_{e+1}})_{h_i \in \mathcal{H}} \cup pk_{e+1}$.
- Return $(pk_{e+1}, sk_{e+1}, \Delta_{e+1})$.

$\mathsf{Sig}(sk_e, m)$:

- Set $\mathcal{H} \leftarrow \mathcal{H} \cup H(m)$.
- Return $\sigma_e \leftarrow H(m)^{sk_e}$.

$\mathsf{UpdateS}(\Delta_{e+1}, \sigma_e, m)$:

- Find $(H(m), \Delta_{H(m),e+1}) \in \Delta_{e+1}$ using $H(m)$; if not found, return 0.
- Return $\sigma_{e+1} \leftarrow \sigma_e \cdot \Delta_{H(m),e+1}$.

$\mathsf{Ver}(pk_e, m, \sigma_e)$:

- Return $e(H(m), pk_e) = e(\sigma_e, \tilde{g})$.

Fig. 4. BLS-USpt.

4.2. Security of BLS-USpt

We first prove that BLS-USpt is USpt-UU-CMA (Definition 10) secure: the adversary is trying to distinguish signatures generated by Sig or UpdateS; we are going to use the fact that the updated signature is the same as the signature generated by the corresponding updated signing key, *e.g.*, by Definition 11. Then, we prove that BLS-USpt is USpt-EUF-CMA (Definition 9) secure: the adversary is trying to forge a valid signature; by introducing a BLS forger, the existence of BLS-USpt could lead to a BLS-signature forgery; therefore, by the security of BLS, we can conclude that BLS-USpt is USpt-EUF-CMA.

4.2.1. USpt-UU-CMA

Formally, the adversary $\mathcal{A}$ is given sig, next, updates, corrupt, and ver oracles, and then it returns a target epoch e^* and a message m^* which it queried $\sigma^*_{e'} \leftarrow sig(m^*)$ in some epoch $e' < e^*$. In game 0, the experiment $\mathsf{Exp}^{\text{uspt-euf-cma}}_{\text{BLS-USpt},\mathcal{A}}$ with $b = 0$ returns $\sigma^{(0)} \leftarrow \mathsf{Sig}(sk_{e^*}, m^*)$ to $\mathcal{A}$. In game 1, the experiment $\mathsf{Exp}^{\text{uspt-euf-cma}}_{\text{BLS-USpt},\mathcal{A}}$ with $b = 1$ returns $\sigma^{(1)} \leftarrow \mathsf{UpdateSCh}(m^*, e', e^*)$ to $\mathcal{A}$. Note that UpdateSCh calls Next to get the signature-update tokens as input, then the adversary $\mathcal{A}$ also accesses the tokens $(\Delta_{e'+1}, \Delta_{e'+2}, \dots, \Delta_{e^*})$. Afterwards, $\mathcal{A}$ returns 0 if $\mathsf{updates}(\Delta_{e'+1}, m^*, \sigma^*_{e'}) = 0$ and 1 otherwise.

Theorem 1. *The* BLS-USpt *from Fig. 4 is USpt-UU-CMA secure.*

Proof. Formally, we consider the following sequence of games:

Game 0. This is the experiment $\mathsf{Exp}^{\text{uspt-euf-cma}}_{\text{BLS-USpt},\mathcal{A}}$ with $b = 0$, where we return $\sigma^{(0)} \leftarrow \mathsf{Sig}(sk_{e^*}, m^*)$.

Game 1. This is the experiment $\mathsf{Exp}^{\text{uspt-euf-cma}}_{\text{BLS-USpt},\mathcal{A}}$ with $b = 1$, where we return $\sigma^{(1)} \leftarrow \mathsf{UpdateSCh}(m^*, e', e^*)$.

Note that the adversary $\mathcal{A}$ is given sig, next, updates, corrupt, and ver oracles, and therefore access to all keys, tokens, and signatures. In **Game 0**, let the key pair in epoch e^* be (sk_{e^*}, pk_{e^*}). In **Game 1**, let the key pair in epoch e' be $(sk_{e'}, pk_{e'})$. We define (sk_{e^*}, pk_{e^*}) for **Game 1** by Next and record all keys and signature-update tokens in the global state set $\mathcal{I}$. Then, algorithm $\mathsf{UpdateSCh}(m^*, e', e^*)$: works as follows: for $i = e', \dots, e^* - 1$, compute $\sigma_{i+1} \leftarrow \mathsf{UpdateS}(\Delta_{i+1}, m^*, \sigma_i)$, where $\sigma_{i+1} = \sigma_i \cdot \Delta_{H(m^*),i+1}$; return σ_{e^*}.

Since BLS-USpt is EFUS, therefore the output of UpdateSCh $\sigma_{e^*} = \sigma^{(1)}$ is identical to the signature $\sigma^{(0)}$. This concludes that $\mathsf{Adv}^{\text{uspt-uu-cma}}_{\text{BLS-USpt},\mathcal{A}}(\lambda, n) = 0$ for any adversary $\mathcal{A}$, which concludes the proof of USpt-UU-CMA security. □

4.2.2. USpt-EUF-CMA

We reduce USpt-EUF-CMA security of BLS-USpt to the EUF-CMA security of BLS[1]. Specifically, we use the Sig oracle of EUF-CMA challenger in epoch e^*. Except for this challenge epoch, we have to answer sig, updates, and corrupt queries, where the last one will leak the signing key to the adversary. Recall that a valid challenge should be formed like $(m^*, \sigma^*_{e^*}, e^*)$, where no sig query on m^* for any epoch before e^* and no corrupt for epoch e^* (Fig. 2).

Except for the challenge epoch e^*, we simulate the original game with knowledge of signing keys and key-update tokens. For the challenge epoch e^*, we have to keep the signature from epoch $e^* - 1$ can be updated and verified by pk_{e^*}. Unlike the scheme, we locally record the signed messages, not their hashes, in the global state set $\mathcal{S}$. When queried to next in epoch $e^* - 1$, for each message $m_i \in \mathcal{S}$, we first query $\mathsf{Sig}(m_i)$ to the EUF-CMA challenger of BLS associated with e^*, then compute signature-update token $\Delta_{e^*} := (H(m_i), \sigma_i/\sigma'_i)_{m_i \in \mathcal{S}} \cup pk^*_e$, where σ_i, σ'_i is the signature of m_i in epoch $e^*, e^* - 1$ respectively. Similarly, when queried to next in epoch e^*, we first generate a fresh key pair (pk_{e^*+1}, sk_{e^*+1}) to sign every $m_i \in \mathcal{S}$ using sk_{e^*+1}, then compute signature-update token $\Delta_{e^*+1} := (H(m_i), \sigma''_i/\sigma_i)_{m_i \in M} \cup pk_{e^*+1}$, where σ''_i is the signature of m_i in epoch $e^* + 1$. Since BLS-USpt is EFUS, the fresh and updated signatures are indistinguishable. If we get a valid forgery $(m^*, \sigma^*_{e^*}, e^*)$ from adversary $\mathcal{A}$, we can output $(m^*, \sigma^*_{e^*})$ as a forgery to EUF-CMA of BLS.

Theorem 2. *The* BLS-USpt *from Fig. 4 is USpt-EUF-CMA secure.*

Proof. Formally, let us prove it by the following sequence of games, where for each game i, the success event is denoted by $S_{\mathcal{A},i}$.

Game 0. This is the original experiment $\mathsf{Exp}^{\text{uspt-euf-cma}}_{\text{BLS-USpt},\mathcal{A}}$. The successful forgery event of $\mathcal{A}$ is denoted by S_0, where the subscript indicates the game. It holds that

$$\Pr[S_{\mathcal{A},0}] = \Pr\left[\mathsf{Exp}^{\text{uspt-euf-cma}}_{\text{BLS-USpt},\mathcal{A}}(\lambda, n) = 1\right] = \mathsf{Adv}^{\text{uspt-euf-cma}}_{\text{BLS-USpt},\mathcal{A}}(\lambda, n). \tag{1}$$

Game 1. This game is identical to **Game 0** except that we guess an epoch $\hat{e}$ for which $\mathcal{A}$ outputs the forgery in epoch $e^* = \hat{e}$; we abort if $\hat{e} \neq e^*$. Assuming n is the maximum of epochs, and it holds that

$$\frac{1}{n^2}\Pr[S_{\mathcal{A},0}] \leq \Pr[S_{\mathcal{A},1}]. \tag{2}$$

Game 2. This game is identical to **Game 1** except that:

- For call to next in epoch $\hat{e} - 1$, we compute $h_i \leftarrow H(m_i), \sigma_{\hat{e},i} \leftarrow \mathsf{BLS.Sig}(sk_{\hat{e}}, m_i)$ for each message $(\hat{e} - 1, m_i, \sigma_{\hat{e}-1,i}) \in \mathcal{S}$, and set $\Delta_{\hat{e}} := (h_i, \sigma_{\hat{e},i}/\sigma_{\hat{e}-1,i})_{m_i \in \mathcal{S}} \cup pk_{\hat{e}}$.
- For call to next in epoch $\hat{e}$, we run $(sk_{\hat{e}+1}, pk_{\hat{e}+1}) \leftarrow \mathsf{BLS.Gen}(\lambda)$ to obtain a fresh key pair for epoch $\hat{e} + 1$. For each message $m_i \in \mathcal{S}$, we compute $h_i \leftarrow H(m_i)$ and run $\sigma_{\hat{e}+1,i} \leftarrow \mathsf{BLS.Sig}(sk_{\hat{e}+1}, m_i)$, and set $\Delta_{\hat{e}+1} := (h_i, \sigma_{\hat{e}+1,i}/\sigma_{\hat{e},i})_{m_i \in \mathcal{S}} \cup pk_{\hat{e}+1}$.
- For each call to sig for message m in epoch $\hat{e}$, we run $\sigma \leftarrow \mathsf{BLS.Sig}(sk_{\hat{e}}, m)$ and add $(\hat{e}, m, \sigma)$ to $\mathcal{S}$.

Now we claim **Game 2** and **Game 1** are indistinguishable; that is, they are distributed identically. We generate the signature-update tokens by knowing the challenge public key and the messages that need to be updated in advance. Note that we do not have to modify the update oracle since the modified tokens $(\Delta_{\hat{e}}, \Delta_{\hat{e}+1})$ can be used for signature valid updates and can be verified by public keys associated to epochs, and the distribution of tokens is identical to the original one. Therefore, it holds that

$$\Pr[S_{\mathcal{A},1}] = \Pr[S_{\mathcal{A},2}]. \tag{3}$$

Now, we claim that we can associate an EUF-CMA challenger for BLS and construct a valid adversary $\mathcal{B}$ for it. In this case, the challenge key pair is $(sk_{\hat{e}}, pk_{\hat{e}})$ where $\mathcal{B}$ only gets knowledge of $pk_{\hat{e}}$, $\mathcal{B}$ also has the access to oracle $\mathsf{BLS.Sig}(sk_{\hat{e}}, \cdot)$. Recall that a valid EUF-CMA adversary $\mathcal{B}$ for BLS is not allowed to query $sig(sk_{\hat{e}}, m^*)$, and $\mathcal{A}$ fails the game if it queried $\mathsf{sig}(m^*, e')$ for message m^* in some epoch $e' \leq \hat{e}$ ($\hat{e} = e^*$) or $\mathsf{corrupt}(e^*)$. Now we need to take care of the situation that $\mathcal{A}$ has all signatures for message m^* for epoch $\hat{e} < e' < n$ and all signing keys except epoch $\hat{e}$ are obtained. We w.l.o.g assume that $\mathcal{A}$ queried $\mathsf{sig}(m^*, \hat{e}+1)$ since all subsequent signatures can be obtained via updates. Note that before m^* is signed, it will not be added to set $\mathcal{S}$. No update can be done since the signature-update token Δ does not include such a pair $(H(m^*), \cdot)$[2]. For the signing keys, keys $sk_{\hat{e}-1}$ and $sk_{\hat{e}+1}$ are generated independently to $sk_{\hat{e}}$, therefore, they bring no advantage on forge $\sigma^*_{\hat{e}}$.

Now, we can claim that $\mathcal{B}$ is sufficient to act as a valid adversary for an EUF-CMA challenger if $\mathcal{A}$ non-trivially wins the game. Therefore,

[1] The BLS scheme can be obtained from modified BLS-based US in Fig. A.7 by deactivating Next and UpdateS algorithms, with some trivial changes to remove epochs. We omit the full description for saving space.

[2] The BLS-USpt is somewhat sesquidirectionally ('sesqui' is Latin for one-and-a-half [25]) updatable: suppose a message is firstly signed in epoch e, one cannot obtain a valid signature in epoch $e' < e$, but it can update the signature to a valid one in epoch $e'' > e$. The US in [4] is bidirectional updatable.

$\mathcal{B}$ can get from $\mathcal{A}$ and output (m^*, σ_e^*) as a forgery to the EUF-CMA challenger of BLS; that is,

$$\Pr[S_{\mathcal{A},2}] \leq \mathsf{Adv}^{\text{euf-cma}}_{\text{BLS},\mathcal{B}}(\lambda). \tag{4}$$

Putting ((1),(2),(3),(4)) all together, it holds that,

$$\mathsf{Adv}^{\text{uspt-euf-cma}}_{\text{BLS-USpt},\mathcal{A}}(\lambda, n) \leq n^2 \mathsf{Adv}^{\text{euf-cma}}_{\text{BLS},\mathcal{B}}(\lambda),$$

which concludes this proof. □

5. Application discussion and future work

5.1. Applications in Web3

First, key rotation is a desirable property for Web3 applications in both security and privacy aspects [26]. Users can refresh their private keys to eliminate potential leakage threats, such as side-channel attacks and phishing websites.

The essence of the US scheme is to update the signature instead of re-issuing it, which is helpful in cases where some signatures have exceptional value in Web3. Specifically, as signatures are published on the blockchain, if there is a need for subsequent continuous verification (stored as data on the chain), our solution can help the owners easily update their keys and signatures. Application scenarios where key rotation can be completed including but not limited to authorship, ownership of digital assets, and copyrights of artworks [27]; data sharing, delegation [28], and auditing [29]; product tracing [30]. Note that the key-rotation property is not helpful for one-shot-verification signatures, *e.g.*, verification for transactions, since those signatures have completed their mission after being verified by miners.

On an abstract level, let smart contracts on blockchain manage the signatures (together with the data or hashes); we consider three scenarios when the signer needs to update the signing key: common solution, the signer retrieves the data and re-initiate a transaction to store the signature signed by the updated key; US solution, the signer encrypts and transfers its signature-update token and the smart contract performs private computation to update the signatures, *e.g.*, ZEXE system [31]; Updatable Signature with public tokens (USpt) solution, the signer computes and transfers the public token and the smart contract performs regular computation to update the signatures. In the common solution, the previous data lost its legitimacy automatically since the newer-uploaded one replaced them, which is undesirable in some traceability and time-sensitive applications. Moreover, the signer may need to initiate transactions equal to the number of signatures to complete the update of all signatures, which will bring great computing and communication consumption. The US solution requires a blockchain that supports private computation to complete the update. It is necessary to prevent attackers from obtaining tokens in the public blockchain and calculating the new signature key, rendering a high on-chain computing cost and unable fitting in the mainstream blockchains. Our solution (USpt) bypasses those difficulties, which means that the signer does not need to re-upload new data entries, and the ledger system does not support private computation, preserving the traceability of data entries and the practicality of the system. Fig. 5 shows a schematic diagram of the solution comparison.

5.2. Future work

This construction of our solution requires the signer to compute and distribute every token of *legitimate* signatures. Here, we refer to *legitimate* signatures as the signatures that the signer, not the adversary, signed. In this construction, suppose there are n_s signatures to be updated, the signer needs to publish a string $\Delta_{h_{m_1},e+1} \| \Delta_{h_{m_2},e+1} \| \cdots \| \Delta_{h_{m_{n_s}},e+1} \| pk_{e+1}$ for an update, which is $n_s + 1$ group elements. Although this string is transferred in the authenticated channel, this is equally to publish the set of newly generated signatures under the updated signing key. The problem is that the signature-update token is linear growth to the number of messages that need to be updated. However, this is not a problem specific to this scheme; this token that relies on legitimate messages requires the ability to identify legitimate messages, and publishing the entire set of legitimate messages is the simplest way. One may consider using a bloom filter to improve the performance, while the probabilistic nature of the bloom filter makes it inappropriate for cryptographic purposes. Considering the aggregate nature of BLS signatures, we lost the signature updatability if the token is aggregated as $(\prod_1^{n_s} \Delta_{h_{mi},e+1}) || pk_{e+1}$. In this case, the agent can detect if there is any signature not sent by the signer in the previous epoch e; probably, the signing key is lost, and a covert attacker made this forgery. Therefore, when we want to reduce the communication cost of updates, we may violate the idea of signature renewal, but it may be helpful in other scenarios. This needs further research.

A popular Web3-style storage is to implement decentralized cloud storage of data through the IPFS protocol. Signature can be used as an ownership of the data asset in this case. However, the data blocks stored by IPFS cannot be changed, and updating is the process of re-publishing using common methods. If the chameleon hash [32] can be combined to implement the update of ciphertext or signature without changing the hash value, the data addressing basis in IPFS will not be changed. This will be an important future work to achieve key updatability in Web3.

USpt scheme allows the adversarial signer to deny some signatures it signed via updates. This is also an intractable problem. The root of this problem is that in our model, we place more trust in the signer than in a trusted third party. After all, the loss of a private (signing) key represents a loss of identity in Web3. The US with secret tokens updates all valid signatures in the previous epoch, no matter the legitimate or adversarial ones, while preserving non-repudiation. We view this difference as a trade-off in trust choice.

CRediT authorship contribution statement

Haotian Yin: Writing – review & editing, Writing – original draft, Validation, Project administration, Methodology, Investigation, Formal analysis, Conceptualization. **Jie Zhang:** Writing – review & editing, Supervision, Methodology, Investigation, Funding acquisition, Conceptualization. **Wanxin Li:** Writing – review & editing, Validation, Supervision, Methodology, Investigation. **Yuji Dong:** Writing – review & editing, Validation, Methodology, Investigation. **Eng Gee Lim:** Writing – review & editing, Supervision. **Dominik Wojtczak:** Supervision, Investigation.

Declaration of competing interest

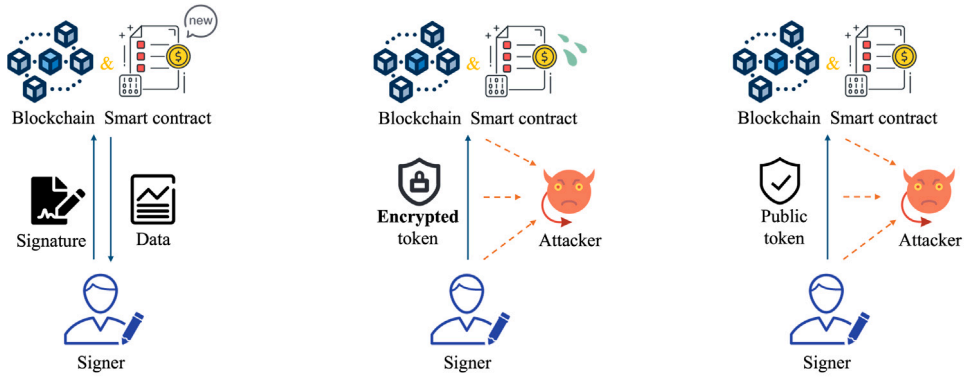
The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was partially supported by the XJTLU Research Development Fund under Grant No. RDF-21-02-014 and RDF-22-02-106; XJTLU Teaching Development Fund under Grant No. TDF2223-R25-207; and Suzhou Municipal Key Laboratory for Intelligent Virtual Engineering (SZS2022004).

Appendix A. Signature-update token and key-update token

We show why USpt cannot be trivially constructed from the Key-Homomorphic signature-based US (KH-based US).



(a) Common solution (b) US solution (c) USpt solution

Fig. 5. Comparison of different solutions on key-rotation of signatures.

$\mathsf{Setup}(\lambda, n)$:

- Return $(pk_1, sk_1) \leftarrow \Sigma.Gen(\lambda)$.

$\mathsf{Next}(pk_e, sk_e)$:

- Choose random $\delta_{e+1} \in \mathbb{H}$.
- Set $\Delta_{e+1} := (\delta_{e+1}, pk_e)$
- Compute $sk_{e+1} \leftarrow \mathsf{UpdateK}(\delta_{e+1}, sk_e)$.
- Compute $pk_{e+1} \leftarrow pk_e \cdot \mu(\delta_{e+1})$.
- Return $(pk_{e+1}, sk_{e+1}, \Delta_{e+1})$.

$\mathsf{Sig}(sk_e, m)$:

- Return $\Sigma.\mathsf{Sig}(sk_e, m)$.

$\mathsf{UpdateS}(\Delta_{e+1}, \sigma_e)$:

- Parse $\Delta_{e+1} = (\delta_{e+1}, pk_e)$.
- Compute $\sigma_{e+1} \leftarrow \Sigma.\mathsf{Adapt}(pk_e, m, \sigma_e, \delta_{e+1})$.
- Return (m, σ_{e+1}).

$\mathsf{Ver}(pk_e, m, \sigma_e)$:

- Return $\Sigma.\mathsf{Ver}(pk_e, m, \sigma_e)$.

Fig. A.6. KH-based US.

A.1. Decoupling Update Tokens

Let Σ be a KH signature, the KH-based US is shown in Fig. A.6. To clarify the difference, we define $\mathsf{UpdateK}(\delta_{e+1}, sk_e) = sk_e + \delta_{e+1}$, which is called as a subroutine in Next.

We found that the Next algorithm first updates the key pair by the key-update token δ_{e+1}, and then outputs the signature-update token Δ_{e+1} which is a tuple including δ_{e+1}. This fact induces a signing key sk_e to be updated by the signature-update token Δ_{e+1} easily. That is, one can compute a *transform function* f such that $f(\Delta_{e+1}) = \delta_{e+1}$. Therefore, no KH-based US is secure by trivially publicizing the signature-update tokens since the adversary can use it to update the signing key and continually forge signatures. We also found this problem is inherent in the KH-based US since all those updatabilities are achieved by the Adapt algorithm (Definition 3), and the security of KH-based US relies on the perfect adaptability (Definition 4). Therefore, we should avoid computing key-update tokens from the signature-update tokens. There are two natural ways to achieve this goal: constructing a USpt without f or constructing a USpt with infeasible computable f. The latter one is a bit more complex; one can understand that f is the inverse of a one-way function f^{-1}. Next, we prove that the first idea is not achievable when the fresh signatures and the updated signatures are identical, and the updatability in the second solution cannot be achieved via the perfect adaptability notion.

A.2. Existence of transform function f

Note that the perfect adaptability implies the signature obtained from Adapt is identical to the signature obtained from Sig given the same message m, key pair (sk_e, pk_e), and tokens $(\delta_{e+1}, \Delta_{e+1})$. We define a notion for US schemes to unveil the equality of fresh and updated signatures (EFUS).

Definition 11 (*Equality of Fresh and Updated Signatures*)**.** A US scheme US is EFUS, if for any epoch $e, e+1 \in [n]$, signing key $sk_e, sk_{e+1} \in \mathcal{SK}$, tokens $\Delta_{e+1} \in \mathcal{ST}, \delta_{e+1} \in \mathcal{KT}$, $r \in \mathcal{R}$, and $m \in \mathcal{M}$, where $\mathcal{SK}, \mathcal{ST}, \mathcal{KT}, \mathcal{R}$ and $\mathcal{M}$ are the sets of signing keys, signature-update tokens, key-update tokens, randomness and messages, such that

$$\mathsf{Sig}(\mathsf{UpdateK}(\delta_{e+1}, sk_e), m; r) = \mathsf{UpdateS}(\Delta_{e+1}, m, \mathsf{Sig}(sk_e, m; r)). \tag{A.1}$$

If the signature scheme is not a unique signature scheme, we obtain different signatures on the same message using different randomness with overwhelming probability. That is why we explicitly define the randomness in the above definition. Now, we are ready to prove the impossibility of constructing an EFUS-US without f.

Lemma 1. *Suppose a US scheme* US *is EFUS. Given any* $m \in \mathcal{M}, r \in \mathcal{R}, sk_e \in \mathcal{SK}$*, we can rewrite the left side of* (A.1) *as*

$$\mathsf{Sig}_{m,r}(\mathsf{UpdateK}_{sk_e}(\delta_{e+1}));$$

then, function f *exists, if functions* $\mathsf{Sig}_{m,r}$ *and* $\mathsf{UpdateK}_{sk_e}$ *are bijective.*

Proof. Because $\mathsf{Sig}_{m,r}$ and $\mathsf{UpdateK}_{sk_e}$ are bijective, therefore they are invertible, and their composite function $g := \mathsf{Sig}_{m,r} \circ \mathsf{UpdateK}_{sk_e}$ is also invertible, the inverse of which is denoted as $g^{-1} : \mathcal{S} \to \mathcal{KT}$. We rewrite the right side of (A.1) as $\mathsf{UpdateS}_{m,r,sk_e}(\Delta_{e+1})$, and denote it as $h(\Delta_{e+1})$ for short, where $h : \mathcal{ST} \to \mathcal{S}$ and we hide $\mathsf{Sig}(sk_e, m; r)$ because after fixing m, r, sk_e, it is also a constant. Therefore, we can construct $f(\Delta_{e+1}) = g^{-1}(h(\Delta_{e+1}))$, where $f : \mathcal{ST} \to \mathcal{S} \to \mathcal{KT}$. Lemma 1 is proved. □

Note that Lemma 1 applies to all eligible US that are in line with the assumption, not only KH-based US. However, it is not necessary to construct f according to the steps in the proof, which may introduce discrete logarithm problems. We show the fact that the transform function f exists in BLS-based US with message-independent updates here.

A.2.1. Transform function f in BLS-based US with message-independent updates

For BLS-based US, the main difference from KH-based US is that the update of the signing key uses multiplication instead of addition in the UpdateK algorithm.

The message-independent updates property allows the signature-update algorithm to be run without inputting messages. We modify the Next algorithm from [4] a little bit, from computing multiplication $sk_{e+1} \leftarrow sk_e \cdot \delta_{e+1}$ to computing modular multiplication $sk_{e+1} \leftarrow sk_e \cdot \delta_{e+1} \bmod p$, for signing key size control and succinct proof.

One can easily verify that BLS-based US with message-independent updates is EFUS.

Corollary 1. *In BLS-US with message-independent updates (Fig. A.7), functions* Sig_m *and* $\mathsf{UpdateK}_{sk}$ *are bijective.*

Proof. We prove it by contradiction.

According to the definition, the function $\mathsf{UpdateK}_{sk}(\delta) := sk \cdot \delta \bmod p$, where $sk, \delta \leftarrow \mathbb{Z}_p^*$, which is typically a prime-order modular multiplication, obviously bijective.

Next, we prove that Sig_m is injective. Given a message m (since there is no randomness used in $\mathsf{Sig}_{m,r}$, we omit r for clarity), suppose there are two different signing keys $sk_1 \neq sk_2$, such that $\mathsf{Sig}_m(sk_1) = \mathsf{Sig}_m(sk_2)$, *e.g.*, $H(m)^{sk_1} = H(m)^{sk_2}$. Since $H : \mathcal{M} \to \mathbb{G}_1$, and every element in $\mathbb{G}_1$ can be represented by g, say $H(M) = g^t$, then,

$$H(m)^{sk_1} = H(m)^{sk_2} \Rightarrow g^{t \cdot sk_1} = g^{t \cdot sk_2} \Rightarrow g^{(t \cdot sk_1) - (t \cdot sk_2)} = 1.$$

Therefore, $(t \cdot sk_1) - (t \cdot sk_2) = 0$, $sk_1 = sk_2$, which contradicts our assumption (note that the computation is not over the cyclic group).

Now we prove that Sig_m is surjective. Informally, for every element in the range of Sig_m, there is always at least one element in the domain corresponding to it. We can derive its surjective property of Sig_m from the correctness of the BLS signature: for every valid signature, there must be a valid public key verifying it; and in addition, every valid public key has a corresponding signing key; therefore, the surjective property of Sig_m follows. □

With the above proof, we can construct the transform function f:

$$f := \mathsf{UpdateK}_{sk_e}^{-1} \circ \mathsf{Sig}_m^{-1} \circ \mathsf{UpdateS}_{sk_e}.$$

Given a signature $\sigma_e = H(m)^{sk_e}$, we can update it to $H(m)^{sk_e \cdot \delta_{e+1}}$ by inputting signature-update token $\Delta_{e+1} = (\delta_{e+1}, pk_e)$ to $\mathsf{UpdateS}_{sk_e}$. Then we can apply Sig_m^{-1} to get $sk_{e+1} = sk_e \cdot \delta_{e+1}$ (which involves the discrete logarithm problems on group $\mathbb{G}_1$). Finally, one can use $\mathsf{UpdateK}_{sk_e}^{-1}(sk_{e+1})$ to get the key-update token δ_{e+1}. Note that Lemma 1 only implies the existence of f, not promising a "feasible" construction. Indeed, computing the signing key from a signature is always a devastating task for the security of a signature scheme.

This fact interestingly implies the possible way to construct a computationally infeasible function f, *e.g.*, making it hard to compute (non-polynomially time computable) the key-update token from a signature-update token or, making f^{-1} *one-way*.

A.3. One-wayness of transform function f^{-1}

A straightforward way is to publish the public-key version of the key-update token. As explained in Lemma 1 of [6], if there is a KH signature scheme such that we can replace the update token δ in the Adapt algorithm (Definition 3) with its public key $\mu(\delta)$, this scheme will not provide UUF-NMA (5) since any adversary can update a signature signed by signing key sk to a forged signature that can be verified by the public key $pk^* = \mu(sk) \cdot \mu(\delta)$ without knowledge of δ.

A key point in the informal proof above is that one can compute the signature-update token from any valid signing key and the updated (next epoch's) public key, *e.g.*, $\mu(\delta) \leftarrow pk^* \cdot \mu(sk)^{-1}$. This relationship is intrinsic for the KH-based US.

We claim this lemma in a more *general* version, where if there is a US scheme with *any public* signature-update token Δ (in the case of KH-based US, $\Delta := \mu(\delta)$) that can update all signatures from one epoch to the next, then the signature scheme is not UUF-NMA secure. This corollary sufficiently excludes the possibility of building USpt from perfect adaptability.

Corollary 2. *A US scheme* US *for which its PPT algorithm* Update *takes a signature-update token* $\Delta_{e+1} \in \mathcal{PK}$ *that still satisfies Definition 7 and there exists an efficiently computable function* $g(pk_e, pk_{e+1}) \to \Delta_{e+1}$*, then* US *is not UUF-NMA-secure.*

Proof. We prove this corollary by constructing an adversary against UUF-NMA security of US. Let $\mathcal{C}$ be an UUF-NMA challenger with input pk_{e+1} and a target message m^*, its goal is to output a valid signature σ_{e+1} such that $\mathsf{Ver}(pk_{e+1}, m, \sigma_{e+1}) = 1$. $\mathcal{C}$ runs $\mathsf{Setup}(\lambda, n)$ to get a key pair (pk_1, sk_1) and sets $(pk_e, sk_e) := (pk_1, sk_1)$, then computes $\sigma_e \leftarrow \mathsf{Sig}(sk_e, m)$, and finally, outputs a forgery $\sigma_{e+1} \leftarrow \mathsf{UpdateS}(g(pk_e, pk_{e+1}), m, \sigma_e)$. The success of forgery follows the correctness of updatable signatures. □

$\mathsf{Setup}(\lambda, n)$:

- Choose three groups $\mathbb{G}_1 = \langle g \rangle, \mathbb{G}_2 = \langle \tilde{g} \rangle$, and $\mathbb{G}_T$ are three groups of prime order p with $\lambda = \log_2 p$; construct a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$; choose a hash function $H : \mathcal{M} \to \mathbb{G}_1$ uniformly at random from hash function family $\{H_k\}_k$.
- Choose $sk \leftarrow \mathbb{Z}_p^*$, and set $pk := \tilde{g}^{sk}$.
- Return $(pk_1, sk_1) \leftarrow (pk, sk)$.

$\mathsf{Next}(pk_e, sk_e)$:

- Choose $\delta_{e+1} \leftarrow \mathbb{Z}_p^*$, and set $\Delta_{e+1} := (\delta_{e+1}, pk_e)$.
- Compute $pk_{e+1} \leftarrow pk_e^{\delta_{e+1}}$.
- $\mathsf{UpdateK}(\delta_{e+1}, sk_e)$: Compute $sk_{e+1} \leftarrow sk_e \cdot \delta_{e+1} \mod p$.
- Return $(pk_{e+1}, sk_{e+1}, \Delta_{e+1})$.

$\mathsf{Sig}(sk_e, m)$:

- Return $\sigma_e := H(m)^{sk_e}$.

$\mathsf{UpdateS}(\Delta_{e+1}, \sigma_e)$:

- Parse $\Delta_{e+1} = (\delta_{e+1}, pk_e)$.
- Compute $\sigma_{e+1} \leftarrow \sigma_e^{\delta_{e+1}}$.
- Return σ_{e+1}.

$\mathsf{Ver}(pk_e, m, \sigma_e)$:

- Return $e(H(m), pk_e) = e(\sigma_e, \tilde{g})$.

Fig. A.7. BLS-based US with message-independent updates.

An implicit condition is that each signature-update token Δ_{e+1} can update *all* signatures generated in epoch e to epoch e+1. In addition, the existence and bijective of f implies that each signature-update token is bonded to each update of epochs. We need to break this paradigm for other possible USpt constructions. Therefore, if the token is only useful for updating the message signed by the signer, then this kind of attack will fail. The problem is to publish a token that makes the verifiers distinguish the signature signed by the signer from the signature signed by the adversary, which is similar to the problem of proving if some *elements* (signatures) are in a *set* (signature signed by the signer).

Data availability

No data was used for the research described in the article.

References

[1] Cohn-Gordon K, Cremers C, Garratt L. On post-compromise security. In: 2016 IEEE 29th computer security foundations symposium. IEEE; 2016, p. 164–78.

[2] LaMacchia B, Lauter K, Mityagin A. Stronger security of authenticated key exchange. In: International conference on provable security. Springer; 2007, p. 1–16.

[3] Krawczyk H. HMQV: A high-performance secure Diffie-Hellman protocol. In: Annual international cryptology conference. Springer; 2005, p. 546–66.

[4] Cini V, Ramacher S, Slamanig D, Striecks C, Tairi E. Updatable signatures and message authentication codes. In: IACR international conference on public-key cryptography. Springer; 2021, p. 691–723.

[5] Canetti R. Universally composable security: A new paradigm for cryptographic protocols. In: Proceedings 42nd IEEE symposium on foundations of computer science. IEEE; 2001, p. 136–45.

[6] Derler D, Slamanig D. Key-homomorphic signatures: definitions and applications to multiparty signatures and non-interactive zero-knowledge. Des Codes Cryptogr 2019;87:1373–413.

[7] Boneh D, Lynn B, Shacham H. Short signatures from the Weil pairing. In: International conference on the theory and application of cryptology and information security. Springer; 2001, p. 514–32.

[8] Lehmann A, Tackmann B. Updatable encryption with post-compromise security. In: Advances in cryptology–EUROCRYPT 2018: 37th annual international conference on the theory and applications of cryptographic techniques, Tel Aviv, Israel, April 29-May 3, 2018 proceedings, part III 37. Springer; 2018, p. 685–716.

[9] Zhou J, Liu Z. An improved updatable signature scheme with weakened token. In: International conference on frontiers in cyber security. Springer; 2023, p. 419–38.

[10] Diffie W, Hellman ME. New directions in cryptography. In: Democratizing cryptography: the work of Whitfield Diffie and Martin Hellman. 2022, p. 365–90.

[11] Barak B, Goldreich O, Impagliazzo R, Rudich S, Sahai A, Vadhan S, et al. On the (im) possibility of obfuscating programs. In: Annual international cryptology conference. Springer; 2001, p. 1–18.

[12] Jain A, Lin H, Sahai A. Indistinguishability obfuscation from well-founded assumptions. In: Proceedings of the 53rd annual ACM SIGACT symposium on theory of computing. 2021, p. 60–73.

[13] Zhou J, Liu Z, Wang B. Updatable signature scheme with weakened token and no-directional key update. Int J Netw Manage 2025;35(1):e2304.

[14] Bellare M, Miner SK. A forward-secure digital signature scheme. In: Annual international cryptology conference. Springer; 1999, p. 431–48.

[15] Beck G, Choudhuri AR, Green M, Jain A, Tiwari PR. Time-deniable signatures. In: Proceedings on Privacy Enhancing Technologies. 2023.

[16] Chatzigiannis P, Chalkias K, Kate A, Mangipudi EV, Minaei M, Mondal M. SoK: Web3 recovery mechanisms. 2023, Cryptology ePrint Archive.

[17] Arora SS, Badrinarayanan S, Raghuraman S, Shirvanian M, Wagner K, Watson G. Avoiding lock outs: Proactive fido account recovery using managerless group signatures. 2022, Cryptology ePrint Archive.
[18] Blackshear S, Chalkias K, Chatzigiannis P, Faizullabhoy R, Khaburzaniya I, Kogias EK, et al. Reactive key-loss protection in blockchains. In: Financial cryptography and data security. FC 2021 international workshops: CoDecFin, deFi, VOTING, and WTSC, virtual event, March 5, 2021, revised selected papers 25. Springer; 2021, p. 431–50.
[19] Boyd C, Davies GT, Gjøsteen K, Jiang Y. Fast and secure updatable encryption. In: Annual international cryptology conference. Springer; 2020, p. 464–93.
[20] Doan TV, Psaras Y, Ott J, Bajpai V. Toward decentralized cloud storage with IPFS: opportunities, challenges, and future considerations. IEEE Internet Comput 2022;26(6):7–15.
[21] Wang Q, Li R, Wang Q, Chen S. Non-fungible token (NFT): Overview, evaluation, opportunities and challenges. 2021, arXiv preprint arXiv:2105.07447.
[22] Derler D, Slamanig D. Key-homomorphic signatures: Definitions and applications to multiparty signatures and non-interactive zero-knowledge. 2016, http://dx.doi.org/10.1007/s10623-018-0535-9, Cryptology ePrint Archive, Paper 2016/792 URL https://eprint.iacr.org/2016/792.
[23] Boneh D, Gentry C, Lynn B, Shacham H, et al. A survey of two signature aggregation techniques. 2003, CryptoBytes.
[24] Boneh D, Gentry C, Lynn B, Shacham H. Aggregate and verifiably encrypted signatures from bilinear maps. In: Advances in cryptology—EUROCRYPT 2003: international conference on the theory and applications of cryptographic techniques, Warsaw, Poland, May 4–8, 2003 proceedings 22. Springer; 2003, p. 416–32.
[25] Poettering B, Rösler P. Towards bidirectional ratcheted key exchange. In: Advances in cryptology–CRYPTO 2018: 38th annual international cryptology conference, Santa Barbara, CA, USA, August 19–23, 2018, proceedings, part i 38. Springer; 2018, p. 3–32.
[26] Torres CF, Willi F, Shinde S. Is your wallet snitching on you? an analysis on the privacy implications of web3. In: 32nd USENIX security symposium. 2023, p. 769–86.
[27] Chen H, Duan H, Abdallah M, Zhu Y, Wen Y, Saddik AE, et al. Web3 metaverse: State-of-the-art and vision. ACM Trans Multimed Comput Commun Appl 2023;20(4):1–42.
[28] Gao H, Duan P, Pan X, Zhang X, Ye K, Zhong Z. Blockchain-enabled supervised secure data sharing and delegation scheme in Web3. 0. J Cloud Comput 2024;13(1):21.
[29] Shen J, Guo F, Chen X, Susilo W. Secure cloud auditing with efficient ownership transfer. In: Computer security–ESORICS 2020: 25th European symposium on research in computer security, ESORICS 2020, Guildford, UK, September 14–18, 2020, proceedings, part I 25. Springer; 2020, p. 611–31.
[30] Assaqty MIS, Gao Y, Hu X, Ning Z, Leung VC, Wen Q, et al. Private-blockchain-based industrial IoT for material and product tracking in smart manufacturing. IEEE Netw 2020;34(5):91–7.
[31] Bowe S, Chiesa A, Green M, Miers I, Mishra P, Wu H. Zexe: Enabling decentralized private computation. In: 2020 IEEE symposium on security and privacy. IEEE; 2020, p. 947–64.
[32] Li Y, Liu S. Tagged chameleon hash from lattices and application to redactable blockchain. 2023, Cryptology ePrint Archive, Paper 2023/774 URL https://eprint.iacr.org/2023/774.