History Binding Signature*

Shlomi Dolev and Matan Liber

Department of Computer Science, Ben-Gurion University of the Negev, Beersheba, Israel
dolev@cs.bgu.ac.il
matanli@post.bgu.ac.il

Abstract. Digital signatures are used to verify the authenticity of digital messages, that is, to know with a high level of certainty, that a digital message was created by a known sender and was not altered in any way. This is usually achieved by using asymmetric cryptography, where a secret key is used by the signer, and the corresponding public key is used by those who wish to verify the signed data. In many use-cases, such as blockchain, the history and order of the signed data, thus the signatures themselves, are important. In blockchains specifically, the threat is forks, where one can double-spend its crypto-currency if one succeeds to publish two valid transactions on two different branches of the chain. We introduce a single private/public key pair signature scheme using verifiable random function, that binds a signer to its signature history. The scheme enforces a single ordered signatures' history using a deterministic verifiable chain of signature functions that also reveals the secret key in case of misbehaviors.

Keywords: Digital Signatures · Verifiable Secret Sharing · Verifiable Random Functions.

1 Introduction

Digital signatures are used in a wide variety of applications, from signing software distributions to verification of online transactions. In some countries, digital signatures even have legal bindings [6]. One of the key features a digital signature scheme provides, is preventing the signer from claiming to not have signed a message, while a valid signature exists [10]. The fact that a signature provides non-repudiation, i.e., cannot be denied to have been created by the signer, indeed increases its value to the verifier. For example, in the case of digital currency, the paid end of a transaction does not want the payer to deny ever signing, moreover participating in the transaction. Yet in some cases, this is not enough, since the signer can claim for the privacy loss of its private key, or in other cases, the verifier may not know the signature was abused in other ways. Blockchain forks

* Partially supported by the Rita Altura Trust Chair in Computer Science, a grant from the Ministry of Science and Technology, Israel & the Japan Science and Technology Agency (JST), and the German Research Funding (DFG, Grant#8767581199)

are a common example, where two versions of a transaction may be published in two different branches of the chain, leading to double-spending. The general case is where one signs a series of sequential data, where not only the origin of the data is to be verified, but also the index of the data in the series. Since there are many such cases where the signer can gain from successfully signing and publishing different data versions for the same index in the series, we introduce punishment for such behavior.

Previous work. In our previous work [5] we discussed a possible weakness in permissioned blockchains. In such blockchains, a relatively small number of players participate in the BFT algorithm [9] used to determine the next transactions to be published on the blockchain. Those players are the permissioned players (referred by us as nodes), whose copies of the public ledger represent the current state of the blockchain. The permissionless players (referred by us as the private users, or simply users) both view and submit transactions only through the nodes.

It is known that Byzantine consensus can be achieved when up to one-third of the nodes are faulty. Since this boundary is well known, an adversary may try to overtake more nodes than the system can tolerate thus putting the blockchain in a harmful position. We discussed a worst-case scenario where the adversary may even completely destroy many of the nodes' copies of the blockchain and leading to the effective loss of the transactions history, resulting in the loss of the balances of the users.

We suggested a solution that enables a trustless restoration of the blockchain in such scenarios. The solution includes users saving their own transactions history (incoming transactions and payment transactions). The users present their history of transactions (which we assume are signed in an undeniable collective signature) to the nodes in case a ledger restoration is needed.

To prevent users from exploiting the lack of information the nodes hold after an attack, and presenting a partial payments history (resulting in the restoration of a balance greater than they previously had), our solution enforced two conditions:

- The transactions are indexed with an increasing order to prevent skipping payments in the presented history.
- Additional information D_i is published alongside every transaction T_i to prevent hiding a suffix of the payments history.

The first condition was enforced by making a transaction valid only if it included index $i + 1$ when the previously approved transaction of the user was of index i. The second condition was enforced by setting the additional information D_i to include a verifiable share of the user's secret key s, used for signing transactions. When a restoration process takes place, any user presenting a transactions history up to an index m is forced to present a corresponding proof P_m, also including a verifiable share of s.

The main idea is that any attempt to hide a suffix of the transactions history results in publishing enough shares of s that enable the secret's reconstruction.

Contribution. We leverage from the same incentive for honesty behavior to make a chain of signatures both sequential and unique. We use verifiable random functions to create a new signature function for every new index of data to sign. The way the signature functions are made makes them both secure and binding, thus, creating a unique sequence after the initial seed is determined.

Using this digital signature we achieve double-spending prevention, which, in comparison to some other schemes such as [1], has the following properties:

- No need for a "centralized" bank, where coin issuing and verification is performed against.
- Furthermore, there is no interactive per-coin "issuing" process at all, and once the initial public key is accepted and published, yet we do rely on the publicity of the transactions.
- This also means that there is no one secret, that if revealed, the whole system breaks (such as the RSA secret of the bank).
- Given correct propagation/publicity of the transactions (as is the case in blockchains, for example), framing can be done by all users.
- A user being framed for dishonesty also loses the privacy of its account completely.
- Although our scheme results in a more "aggressive" outcome for a double-spender, an honest user is not affected at all, and in an eco-system where this possible penalty is known to all participants, we can expect fewer attempts of undesired behavior.

The rest of the paper is structured as follows: In Section 2, we briefly describe verifiable secret sharing, verifiable secret public sharing (our variation of VSS that was introduced in our previous work), and verifiable random function, which are our building blocks for our new scheme. In Section 3, we introduce our signature scheme, defining the key generation, signature, and verification algorithms, as well as discuss the advantages of the scheme. In Section 4, we list the conditions a signature scheme must meet, including unforgeability, security, and correctness (both of the signature scheme itself and of the key-revealing property of our scheme) and prove our scheme indeed meets them. Finally, in Section 5, we conclude the scheme introduced in this paper and its possible usage, as well as shortly discuss possible future lines of work.

2 Preliminaries

We describe the protocols of secret sharing and verifiable secret sharing that our scheme is based on, as well as our variation of those protocols. We also describe verifiable random functions, which we later incorporate with our previous work to get our result.

2.1 Verifiable Secret Sharing

Shamir [13] introduced Shamir's secret sharing (SSS) as a method to divide a secret into parts (shares), which are generated by the secret owner (dealer), and distributed amongst a group of participants (shareholders). In a (t, n)-threshold (t-out-of-n) secret sharing scheme, n participants, each holding a share, can reconstruct the secret only if t or more of them combine their shares. Moreover, any group of strictly less than t shareholders (including the individual shareholders themselves), learn nothing about the secret.

Verifiable secret sharing (VSS) was introduced by Chor, Goldwasser, Micali and Awerbuch [2], as a secret sharing scheme where every participant can verify that the secret shares are consistent. This is important for preventing a malicious dealer from sending shares to different participants, which do not define a (single) secret.

A (t, n)-threshold secret sharing scheme, consists of a probabilistic polynomial-time algorithm (PPTA) Share_G and a polynomial-time algorithm (PTA) $\mathit{Recover}_G$, for some global parameters G. The global parameters G are clear from the context so we drop G from the notation. The algorithm $\mathit{Share}(s) \rightarrow \{(1, s_1), (2, s_2), \dots, (n, s_n)\} = S(s)$ takes a secret key s as an input, and outputs n shares $(1, s_1), (2, s_2), \dots, (j, s_j), \dots, (n, s_n)$ where j is the share's index and s_j is the share's value. The algorithms $\mathit{Recover}((a_1, s_{a_1}), (a_2, s_{a_2}), \dots, (a_t, s_{a_t})) \rightarrow s$ takes as an input any t valid distinct shares with share indices $\{a_1, \dots, a_t\} \subseteq [1, n]$, and outputs the original secret s. Formally,

$$\forall s. \mathit{Share}(s) \rightarrow S(s) \implies \forall T' \in \{T \subseteq S(s) | \, |T| = t\}, \mathit{Recover}(T') = s$$

To make this scheme verifiable, we introduce two additional PTAs. *Commit*$(c) \rightarrow C$ that takes a random coefficient c generated by the user and outputs a commit C for it, and $\mathit{Verify}(s(i), C_1, \dots, C_{t-1}, y) \rightarrow res \in \{ACCEPT, REJECT\}$ that takes a share, commits for both of the polynomial's coefficients, and the public key of the user, and ACCEPTs if the share is valid or REJECTs otherwise.

We use Feldman's [7] verifiable secret sharing scheme to define:

- $\mathit{Share}(s) = \{(i, \mathit{Pol}(i) \; mod \; q) | 1 \leq i \leq n\}$ where $\mathit{Pol}(x) = s + \sum_{j=1}^{t-1} c_j x^j$ for some random coefficients $0 < c_j < q$.
- $\mathit{Recover}((a_1, s_{a_1}), \dots, (a_t, s_{a_t})) = s$ using polynomial interpolation.
- $\mathit{Commit}(c) = g^c \; mod \; p$
- $\mathit{Verify}((i, s_i), C_1, \dots, C_{t-1}, y) = \text{ACCEPT} \iff g^{s_i} \; mod \; p = g^{s + \sum_{j=1}^{t-1} c_j i^j \; mod \; q} \; mod \; p = g^s \cdot \mathop{\mathit{\Pi}}_{j=1}^{t-1} (g^{c_j})^{i^j} \; mod \; p = y \cdot \mathop{\mathit{\Pi}}_{j=1}^{t-1} C_j^{i^j} \; mod \; p$

Where p is prime, q is prime divisor of $(p - 1)$, g is a generator of a subgroup of order q in the multiplicative group of $\mathbb{Z}_p^*$, such that $1 < g < p$ and the global parameters G are p, q and g.

2.2 Verifiable Secret Public Sharing

In the classic secret sharing scenario, one generates shares of the secret s and deals different shares to different parties. This implies that right away all of the shares are in the hands of the parties, where each party holds only some of the shares. Later, when reconstruction is required, the parties combine the shares and reveal the secret.

In [5] we introduced the concept of Verifiable Secret Public Sharing (VSPS), i.e., publicly publishing some (verifiable) shares of the secret. In our protocol for enforcing the reveal of the current balance of a crypto-currency wallet, one gradually publishes shares of the secret key s used for signing transactions. Restoring the balance of the wallet also involves publishing a share of s. The key idea is that for every transaction a new SSS polynomial is used, and an honest user does not surpass the threshold for any single SSS instance (thus not revealing s). On the other hand, a dishonest user is forced to publish enough shares corresponding to one of the polynomials, so that the threshold is met, and anyone can restore the signature key s, thus essentially stealing the crypto-currency wallet.

As mentioned in Section 1, we proposed adding additional information D_i for every transaction, containing some of the secret shares of s regarding a new polynomial $Pol_i(x)$ (as well as a share of $Pol_{i-1}(x)$). In addition, balance restoration involves publishing a proof $P_{m'}$, corresponding to the claimed latest m'^{th} transaction. This proof contains an additional share, thus, resulting in the surpassing of the threshold number of published shares needed for restoring s, corresponding to $Pol_{m'}(x)$ in case D_m was published in the past for some $m > m'$.

The suggested structure for such D_i, P_i is:

- $D_i = (s_i(1), s_{i-1}(2), C_{i1}, C_{i2}, C_{(i-1)1}, C_{(i-1)2})$ [1]
- $P_i = (s_i(v), C_{i1}, C_{i2})$, for a random $2 < v \leq q - 1$.

In this case, we used a $(3, 3)$-threshold verifiable secret sharing scheme, meaning $Pol_i(x) = s + c_{i1}x + c_{i2}x^2$, for some random coefficients $0 < c_{i1}, c_{i2} < q$.

One may notice that if the published transactions history is $\bar{H}(m) = \{D_i | 1 \leq i \leq m\}$, then the total published shares are $\{s_i(1), s_i(2) | 1 \leq i \leq m-1\} \cup \{s_m(1)\}$. This means that publishing $P_{m'}$ results in exposing three shares of $Pol_{m'}(x)$ if $m' < m$. Yet this publication keeps the number of exposed shares corresponding $Pol_i(x)$ under the scheme's threshold for every $1 \leq i \leq m$, if $m' = m$, thus keeping s safe.

2.3 Verifiable Random Functions

Verifiable random functions were introduced by Micali, Rabin, and Vadhan [12]. Using a verifiable random function (VRF), given an input x, the holder of a secret key s can compute the value of the pseudo-random function $\mathcal{F}_s(x)$ and a proof $\mathbf{p}_s(x)$.

Using the proof and the public key $y = g^s$ everyone can check that the value $x' = \mathcal{F}_s(x)$ was indeed computed correctly, yet this information cannot be used to find the secret key s.

An implementation by Yevgeniy Dodis and Aleksandr Yampolskiy [4] defines:

[1] $D_1 = (s_1(1), C_{11}, C_{12})$

- $\mathcal{F}_s(x) = e(g,g)^{\frac{1}{x+s}}$
- $\mathbf{p}_s(x) = g^{\frac{1}{x+s}}$

Where $e(\cdot,\cdot)$ is a bilinear map.

Verification of $\mathcal{F}_s(x)$ is done by checking that:

1. $e(g^x y, \mathbf{p}_s(x)) = e(g,g)^{(x+s)\cdot\frac{1}{x+s}} = e(g,g)$
2. $e(g, \mathbf{p}_s(x)) = e(g,g)^{1\cdot\frac{1}{x+s}} = \mathcal{F}_s(x)$

One may notice that x itself is not used, only in an encrypted form in the verification process, namely, it is only a function of g^x and $\mathbf{p}_s(x)$.

3 History Binding Signature

A history binding signature scheme should bind a signer to the previously published signatures created in the past. Such a scheme is characterized by having the following properties:

- A unique per-index signature function.
- Enabling verification that a message was signed by the signature function corresponding to the claimed signature index.
- Holding two signatures of different messages $V \neq V'$ that were signed using the same signature function can be used to expose the signer's secret key.
- Enabling verification that a signature will enable exposing the secret key of the signer in case of misbehavior.

In general, a history binding signature scheme is defined by the *Key_Generation*, *Signing*$_i$ and *Verify*$_i$ algorithms.

- *Key_Generation*(1^n) randomly generates a private/public key pair $(\mathrm{SK}, \mathrm{PK})$.
- *Signing*$_i(m)$ computes:
  1. A signature $s_i(m)$ on the message.
  2. A commitment C_i indicating the signature enables the exposure of the secret key in case of misbehavior.
  3. A non-interactive zero knowledge proof $\mathbf{p}_{\mathrm{SK}}$ that the signature was computed correctly.
  4. For a total signature of *Sign*$_i(m) = (s_i(m), C, \mathbf{p}_{\mathrm{SK}})$.
- *Verify*$_i(m', (s'_i, C, \mathbf{p}'))$ returns ACCEPT if and only if the following checks pass:
  1. The zero knowledge proof verification.
  2. The commitment verification.

Verifiable Secret Public Sharing as a Signature. We use a similar mathematical technique as the one used in VSPS, but for a different usage - enforcing a unique sequential signature history. In other words, we prevent publishing a signature for some message V_i', where a signature for a different message $V_i \neq V_i'$ was previously published. To achieve our desired goal we use a variant of the additional information D_i from our VSPS as a digital signature for V_i, and describe how the *Key_Generation*, *Signing*$_i$ and *Verify*$_i$ algorithms are to be implemented.

- *Key_Generation*(1^n) randomly generates:
  1. A secret key $\text{SK} = (s, c_0) \in_R \mathbb{Z}_q^* \times \mathbb{Z}_q^*$.
  2. A corresponding public key $\text{PK} = (y, C_0) = (g^s, g^{c_0}) \in \langle g \rangle \times \langle g \rangle$.
- *Signing*$_i(V_i)$ computes:
  1. The new random coefficient $c_i = \mathcal{F}_s(c_{i-1})$ and the corresponding new polynomial $Pol_i(x) = s + c_i x = s + \mathcal{F}_s(c_{i-1})x$.
  2. The proof for the new coefficient $\mathbf{p}_s(c_{i-1})$.
  3. The commitments for the previous and new coefficients $C_i = g^{c_i} = g^{\mathcal{F}_s(c_{i-1}) \bmod q}, C_{i-1} = g^{c_{i-1}}$.
  4. The corresponding share $s_i^{\mathcal{H}}(V_i) = Pol_i(\mathcal{H}(V_i)) \bmod q$.
  5. For a total signature $Sign_i(V_i) = (s_i^{\mathcal{H}}(V_i), C_i, C_{i-1}, \mathbf{p}_s(c_{i-1}))$.
- *Verify*$_i(V_i', (s_i', C_i', C_{i-1}', \mathbf{p}'))$ returns ACCEPT if and only if the following checks pass:
  1. $e(C_{i-1}' y, \mathbf{p}') \overset{?}{=} e(g, g)$.
  2. $g^{e(g,\mathbf{p}') \bmod q} \overset{?}{=} C_i'$.
  3. $g^{s_i'} \bmod p \overset{?}{=} y \cdot {C'}_i^{\mathcal{H}(V_i')} \bmod p$.

  where $\mathcal{H}(\cdot)$ is a publicly-known collision resistant hash function.

When looking at our general definition we may notice that this implementation defines:

- $(\text{SK}, \text{PK}) = ((s, c_0), (y, C_0))$
- $s_i(m) = s_i^{\mathcal{H}}(m)$
- $C = (C_i, C_{i-1})$
- $\mathbf{p}_{\text{SK}} = \mathbf{p}_s(c_{i-1})$
- The first and second checks as the zero-knowledge proof verification.
- The third check as the commitment verification.

Advantages. One may look at D_i, as suggested in our previous scheme, as a digital signature on the current transaction. Although D_i did not incorporate the data of the actual transaction T_i, it could easily be changed to do so. This can be done by evaluating the polynomials $Pol_i(x)$ in $\mathcal{H}(T_i)$ instead of some constant number such as 1 or 2. This, however, is not sufficient, since every new polynomial is defined by random coefficients, that are up to the user to decide. To link a polynomial to a predefined index, one can use multiple pre-ordered secret/public key shares [8]. Using VRF, we replace the method for creating the next polynomial from a random one to a deterministic (and verifiable) one

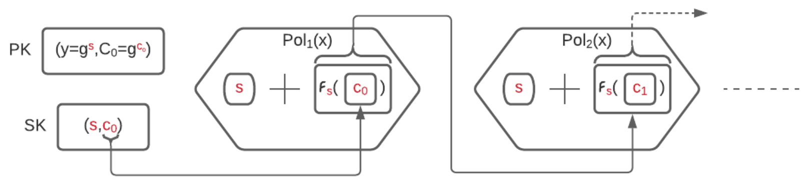

Fig. 1. By using the previous polynomial coefficient as the input for the verifiable random function, we get a unique chain of signature functions.

and achieve the enforcement of the order of the signatures with a single key pair. The fact that there can only be one valid $Pol_i(x)$ per participant (enforced using $\mathcal{F}_s(\cdot)$) replaces the verification the nodes perform (by keeping track of the user's transaction index).

4 Conditions for a Valid Signature

We list the conditions for a valid signature scheme and then prove our scheme meets them:

(a) Unforgeability: for every $i \in \mathbb{N}$ and for every message V to sign, only the holder of SK can generate a valid signature $Sign_i(V)$.
(b) Security: by viewing $\bar{H}(m) = \{Sign_i(V_i) | 1 \leq i \leq m\}$ for any $m \in \mathbb{N}$, one does not learn additional information on SK.
(c) Correctness (signing): $Verify_i(V_i, Sign_i(V_i))$=ACCEPT.
(d) Correctness (key-revealing): if one views $H(m) = \bar{H}(m) \cup Sign_{m'}(V'_{m'})$ for some $m' < m$ and $V'_{m'} \neq V_{m'}$ such that $Sign_{m'}(V_{m'}) \in \bar{H}(m)$, then it can recover the secret signing key and forge valid signatures.

4.1 Unforgeability

Lemma 1. *For every $i \in \mathbb{N}$ and for every message V to sign, only the holder of* SK *can generate a signature $Sign_i(V')$, such that $Verify_i(V', Sign_i(V))$=ACCEPT.*

Proof. We point out that if one tries to forge a signature, i.e., generate a valid signature $Sign'_i(V')= (s'_i, C'_i, C'_{i-1}, \mathbf{p}')$ for a message V', without knowing SK, it must pass all three checks of $Verify_i()$. The first check is identical to the first one in the VRF verification process. In the second check, instead of verifying that $e(g, \mathbf{p}_s(c_{i-1})) = \mathcal{F}_s(c_{i-1})$, we verify that $g^{e(g, \mathbf{p}_s(c_{i-1})) \ mod \ q} = g^{\mathcal{F}_s(c_{i-1}) \ mod \ q}$. One may notice that the value $\mathcal{F}_s(c_{i-1}) = c_i$ is never published, yet $C_i = g^{\mathcal{F}_s(x) \ mod \ q}$ is enough for this verification. The purpose of the third check is to verify the validity of the share, which is the signature, using Feldman's scheme. This check also ensures that the signature contributes to the exposure of the secret key SK in case of misbehaviors. For one to pass check 1 and 2, it would mean breaking [4], and relying on this VRF scheme's own unforgeability, we declare ours is safe as well. □

Man-in-the-Middle Attack. In order to successfully execute such an attack, one needs to generate $s_i^{\mathcal{H}}(V_i')$ for some $V_i' \neq V_i$. Under the assumption that $\mathcal{H}$ is second-preimage resistant, such an attack requires evaluating the polynomial $Pol_i(x)$ in a new point $\mathcal{H}(V_i') \neq \mathcal{H}(V_i)$.

Lemma 2. *An adversary that views only $Pol_i(V)$ for some $i \in \mathbb{N}$, where $Pol_i(0) = s$, cannot evaluate $Pol_i(V')$, for any $V' \neq V$.*

Proof. Assume, for the sake of contradiction, that there exists $v' \neq v$, s.t., an adversary can evaluate $Pol_i(v')$. For a $(2, n)$-threshold, where $n = max(v, v')$ (w.l.o.g $n \geq 2$) SSS, that uses $Pol_i(x)$ as random polynomial for the shares generation, a party that holds the v^{th} share $Pol_i(v)$, generate $Pol_i(v')$. This party can invoke *Recover* with $Pol_i(v)$ and $Pol_i(v')$ as inputs, and reconstruct s, they can break the scheme, although it is a single party (strictly less than 2). It is a contradiction since we know SSS is information-theoretically secure [13]. □

4.2 Security

We analyze the security of the scheme by going over the published values, a signer, with the secret signing key SK, publishes.

Lemma 3. *For any $m \in \mathbb{N}$ and for every $V_1, \ldots, V_m$, a user that publishes $\bar{H}(m)$ does not reveal any information about the secret key* SK.

Proof. If an honest signer created a sequential signature history on the messages $V_1, \ldots, V_m$ for some $m \in \mathbb{N}$, the following values were published:

- The public key PK$= (g^s, g^{c_0})$
- The commitments $C_i = g^{c_i}$ for the coefficients of $Pol_i(x)$, $1 \leq i \leq m$.
- One share $s + c_i \cdot \mathcal{H}(V_i)$ of each polynomial $Pol_i(x)$, $1 \leq i \leq m$.

Under the *Discrete Logarithm Problem* (DLP) [11,3], we may assume PK and the commitments C_i, $1 \leq i \leq m$ give no additional information on s, c_0, c_i, $1 \leq i \leq m$ accordingly. The shares $\{s + c_i \cdot \mathcal{H}(V_i) | 1 \leq i \leq m\}$ each correspond to a different polynomial $Pol_i(x)$. Since no two shares of a single polynomial are published, s cannot be reconstructed using *Recover*. The use of multiple secret sharing polynomials gives the adversary a view of the form $Mx = b$ for:

$$M = \underbrace{\begin{pmatrix} 1 & \mathcal{H}(V_1) & 0 & \ldots & 0 & 0 \\ 1 & 0 & \mathcal{H}(V_2) & \ldots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & 0 & 0 & \ldots & \mathcal{H}(V_{m-1}) & 0 \\ 1 & 0 & 0 & \ldots & 0 & \mathcal{H}(V_m) \end{pmatrix}}_{m \times m+1} x = \underbrace{\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \\ x_{m+1} \end{pmatrix}}_{m+1 \times 1} b = \underbrace{\begin{pmatrix} s_1^{\mathcal{H}}(V_1) \\ s_2^{\mathcal{H}}(V_2) \\ \vdots \\ s_{m-1}^{\mathcal{H}}(V_{m-1}) \\ s_m^{\mathcal{H}}(V_m) \end{pmatrix}}_{m \times 1}$$

Since $|Rank(M)| = m$ but this system represents $m+1$ equations, one cannot distinguish between the real solution $x = \begin{pmatrix} s \\ c_1 \\ \vdots \\ c_{m-1} \\ c_m \end{pmatrix}_{m+1\times 1}$, and the other $q-1$ solutions.

□

4.3 Correctness (signing)

Lemma 4. *For every V_i and a valid signature $Sign_i(V_i)$, $Verify_i(V_i, Sign_i(V_i))$ can be computed and accepts.*

Proof. We defined our $Verify_i(V_i', (s_i', C_i', C_{i-1}', \mathbf{p}'))$ to return ACCEPT if and only if the following checks pass:

1. $e(C_{i-1}' y, \mathbf{p}') \overset{?}{=} e(g, g)$.
2. $g^{e(g,\mathbf{p}')\ mod\ q} \overset{?}{=} C_i'$.
3. $g^{s_i'}\ mod\ p \overset{?}{=} y \cdot {C'}_i^{\mathcal{H}(V_i')}\ mod\ p$.

If an honest user, that holds SK and uses $Sign_i(V_i)$ to sign V_i, then:

- $V_i' = V_i$.
- $s_i' = s_i^{\mathcal{H}}(V_i)$
- $C_i' = C_i = g^{c_i}$.
- $C_{i-1}' = C_{i-1} = g^{c_{i-1}}$.
- $\mathbf{p}' = \mathbf{p}_s(c_{i-1})$.

Then all 3 checks of $Verify_i$ pass since:

1. $e(C_{i-1} y, \mathbf{p}_s(c_{i-1})) = e(g^{c_{i-1}} g^s, \mathbf{p}_s(c_{i-1})) = e(g, g)$.
2. $g^{e(g,\mathbf{p}_s(c_{i-1}))\ mod\ q} = g^{\mathcal{F}_s(c_{i-1})\ mod\ q} = g^{c_i} = C_i$.
3. $g^{s_i^{\mathcal{H}}(V_i)}\ mod\ p = g^{s + c_i \mathcal{H}(V_i)\ mod\ q} = g^s \cdot (g^{c_i})^{\mathcal{H}(V_i)} = y \cdot C_i^{\mathcal{H}(V_i)}\ mod\ p$.

We notice that conditions 1 and 2 can be verified since $y, C_{i-1}, \mathbf{p}_s(c_{i-1})$ and C_i are published and condition 3 can be verified since $s_i^{\mathcal{H}}(V_i)$ is published as well (and we recall that g, p, q and $\mathcal{H}(\cdot)$ are public parameters). □

4.4 Correctness (key-revealing)

We built our scheme in order to prevent a signer from releasing two signatures, corresponding to the same signing index, for two different messages $V \neq V'$. We shall now describe such a scenario and show that this leads to the exposure of the signer's secret key SK.

Lemma 5. *A dishonest user that publishes $H(m) = \bar{H}(m) \cup Sign_{m'}(V_{m'}')$ for some $m' < m$ and $V_{m'}' \neq V_{m'}$ such that $Sign_{m'}(V_{m'}) \in \bar{H}(m)$ enables the exposure of SK and the forgery of valid signatures on its behalf.*

Proof. Let us assume a signer, holding a secret signing key SK, published a series of sequential signatures $\bar{H}(m) = \{Sign_i(V_i)|1 \leq i \leq m\}$ for some $m \in \mathbb{N}$. This means that $\{s_i^{\mathcal{H}}(V_i)|1 \leq i \leq m\}$ was published. If the signer publishes a different signature $Sign_{m'}(V'_{m'})$ for some $m' < m$ and $V'_{m'} \neq V_{m'}$ such that $Sign_{m'}(V_{m'}) \in \bar{H}(m)$, this results in the publication of $s_{m'}^{\mathcal{H}}(V'_{m'})$ as well. This means that regarding $Pol_{m'}(x)$, there are two different published shares $s_{m'}^{\mathcal{H}}(V'_{m'}) \neq s_{m'}^{\mathcal{H}}(V_{m'})$ (due to the fact that $V'_{m'} \neq V_{m'}$ and $\mathcal{H}(\cdot)$ is collision resistant).

Those two shares can be used using SSS *Recover* to find $Pol_{m'}(x) = s + c_{m'}x$. By holding those two values (s and $c_{m'}$), one can recover $c_m = \mathcal{F}_s(c_m) = \mathcal{F}_s(\mathcal{F}_s(c_{m-1})) = \ldots = \mathcal{F}_s^{m-m'}(c_{m'})$. This is sufficient to generate $c_{m+1} = \mathcal{F}_s(c_m)$ and together with the value of s, define $Pol_{m+1}(x) = s + c_{m+1}x$. We notice this also enables one to generate a corresponding proof $\mathbf{p}_s(c_m)$, and in total begin signing on behalf of the original owner of SK.

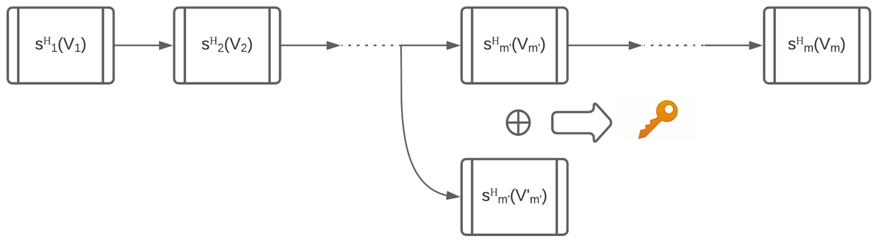

Fig. 2. Publishing a signature corresponding to a different value $V'_{m'}$ where a signature for V_m was already published results in the exposure of s.

We notice that our scheme uses [7] idea for generating verifiable shares (signatures). This prevents one from publishing a signature $Sign_i(V_i)$ that passes our suggested $Verify_i$ but does not enable the others to sign on its behalf in case a different $Sign_i(V'_i)$ is published. □

5 Conclusion and Future Work

We introduced a digital signature scheme based on verifiable secret sharing, where a share of the secret acts as the signature. We used verifiable random functions to create a random, but verifiable, chain of signature functions, each corresponding to an index in the signing chain. This property enables the scheme to provide additional incentive for honesty behavior, by exposing the secret key of the signer if the chain of signatures is abusively forked. This scheme may be used in blockchains, to prevent forks in the public ledger and generally in scenarios where the reliability of the source and uniqueness of versions of data are required.

References

1. Chaum, D., Fiat, A., Naor, M.: Untraceable electronic cash. In: Conference on the Theory and Application of Cryptography. pp. 319–327. Springer (1988)
2. Chor, B., Goldwasser, S., Micali, S., Awerbuch, B.: Verifiable secret sharing and achieving simultaneity in the presence of faults. In: 26th Annual Symposium on Foundations of Computer Science (sfcs 1985). pp. 383–395. IEEE (1985)
3. Coppersmith, D., Odlzyko, A.M., Schroeppel, R.: Discrete logarithms ingf (p). Algorithmica **1**(1-4), 1–15 (1986)
4. Dodis, Y., Yampolskiy, A.: A verifiable random function with short proofs and keys. In: International Workshop on Public Key Cryptography. pp. 416–431. Springer (2005)
5. Dolev, S., Liber, M.: Toward self-stabilizing blockchain, reconstructing totally erased blockchain (preliminary version). In: International Symposium on Cyber Security Cryptography and Machine Learning. pp. 175–192. Springer (2020)
6. Dumortier, J.: Regulation (eu) no 910/2014 on electronic identification and trust services for electronic transactions in the internal market (eidas regulation). In: EU Regulation of E-Commerce. Edward Elgar Publishing (2017)
7. Feldman, P.: A practical scheme for non-interactive verifiable secret sharing. In: 28th Annual Symposium on Foundations of Computer Science (sfcs 1987). pp. 427–438. IEEE (1987)
8. Kubiak, P., Kutyłowski, M.: Preventing a fork in a blockchain – david fighting goliath. In: Proceedings of the IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom 2020). IEEE (2020)
9. Lamport, L., Shostak, R., Pease, M.: The byzantine generals problem. ACM Transactions on Programming Languages and Systems **4**, 382–401 (1982)
10. McCullagh, A., Caelli, W.: Non-repudiation in the digital environment (2000)
11. McCurley, K.S.: The discrete logarithm problem. In: Proc. of Symp. in Applied Math. vol. 42, pp. 49–74. USA (1990)
12. Micali, S., Rabin, M., Vadhan, S.: Verifiable random functions. In: 40th annual symposium on foundations of computer science (cat. No. 99CB37039). pp. 120–130. IEEE (1999)
13. Shamir, A.: How to share a secret. Communications of the ACM **22**(11), 612–613 (1979)