PRIMES Final Research Paper

Signature Scheme with Access Control

Yavor Litchev, Mentor: Yu Xia

September 2021

Abstract

A wide variety of digital signature schemes currently exist, from RSA to El-Gamal to Schnorr. More recently, multi-party signature schemes have been developed, including distributed signature schemes and threshold signature schemes. In particular, threshold signature schemes provide useful functionality, in that they require the number of participating parties to pass a threshold in order to generate a valid signature. However, they are limited in their complexity, as they can only model a threshold function. The proposed signature scheme (monotonic signature scheme) allows for the modeling of complex functions, so long as they are monotonic. This would allow for a much greater degree of access control, all while security and correctness are preserved.

1 Introduction

Digital signatures are integral to cybersecurity. They are an important area in cryptography, and can be used for the verification of messages, financial transactions, data, etc. They provide a practical way for a party to sign messages in an efficient manner using a private key. Afterwards, the message, signature, and public key may be published for public viewing, and any third party may verify the signature using the public key and be highly confident that the message was only signed by the party with the private key. This allows for a robust scheme for authentication and verification.

Many signature algorithms currently exist, including RSA, El-Gamal, etc. Additionally, randomized signature algorithms such as Schnorr provide an added layer of security through the randomization of the signature, irrespective of the message. These algorithms are very practical, however the generation of signatures is limited to only one party. What if one wished to craft a group signature that used multiple participants? There are a number of existing algorithms that seek to solve this problem. One such algorithm is a distributed signature algorithm. Such an algorithm has a set of parties and an access structure. Only for some specific combinations of parties that are validated by the access structure can the parties create a combined signature through their own private keys. Another such algorithm is the threshold signature algorithm, which has an access structure that requires a certain number of parties to pass a threshold. Once the threshold is reached, a valid combined signature may be produced, which can then be verified by a grandmaster public key.

Although these algorithms are highly functional, they are not easily generalizable to more complex functions. One such function is a weighted threshold scheme, where each individual participant has a weight. This could theoretically be solved by assigning multiple private keys to each participant and running a threshold signature scheme, but if complicated weights are used (i.e. rational numbers with large numerators/denominators), such a scheme falls apart as huge numbers of private keys must be generated. If irrational weights are used, then it is impossible to perfectly simulate the correct threshold by assigning multiple keys. Thus, we seek to form a more generalizable access structure. More specifically, we wish to form a signature scheme for multiple parties with potentially any access structure such that making a party participate in the scheme that was previously not participating would never decrease the chances of a valid group signature. We wish to call such a scheme a monotonic signature scheme, and we explore how to craft such a scheme in this paper.

2 Background

We seek to craft a monotonic signature scheme, and in order to do so we need a baseline signature scheme that has key homomorphic properties. This property is significant, since later on we wish to craft a multisignature scheme by splitting up the private key of the baseline signature scheme. Each party will then obtain a piece of the private key. Afterwards, each party will craft their own individual signatures by signing the same message, and using the homomorphic properties, a valid signature may be reconstructed with the shares such that it is equivalent to the signature formed by the original private key. In particular, the algorithm will use AND and OR gates, as well as FANOUT. One such scheme that may be used is the BLS (Boneh–Lynn–Shacham) signature scheme [BLS01]. Other signature schemes with key homomorphism properties may be used, but let us first describe BLS.

Firstly, let us state concepts that will be necessary to describe the Boneh–Lynn–Shacham (BLS) signature scheme. BLS is a deterministic signature scheme with key homomorphism properties. It uses bilinear groups to trivialize the Decisional Diffie Hellman problem, but the Computational Diffie Hellman Problem remains infeasible. The secuirty of the Diffie Hellman protocol is based on how hard it is to compute the discrete logarithm function [BS20]. Let us now define the discrete logarithm problem:

Definition 1 (Discrete Logarithm Problem)**.** Given a group $G = \mathbb{Z}_p$ for some prime p, the discrete log problem is to find a number x such that $g^x = a \mod p$, where $a, g \in \mathbb{Z}_p$.

The above problem is assumed to be computationally infeasible. This provides a very useful one way function, where it is simple to exponentiate but difficult to perform the inverse operation. Let us now introduce the Decisional Diffie Hellman problem and the Computational Diffie Hellman Problem:

Definition 2 (Computational Diffie-Hellman (CDH))**.** Given a group $\mathbb{G} = \mathbb{Z}_p$ for some prime p, and given g^a, g^b for some $g, a, b \in \mathbb{Z}_p$, compute g^{ab}.

Note that CDH is computationally infeasible, as in order to compute g^{ab}, one would require either a or b (as $g^{ab} = (g^a)^b = (g^b)^a$) to solve the discreet log problem. This is a very important property that will be useful in the BLS scheme.

Definition 3 (Decisional Diffie-Hellman (DDH))**.** Given a group $\mathbb{G} = \mathbb{Z}_p$ for some prime p, and given g^a, g^b, g^c for some $g, a, b, c \in \mathbb{Z}_p$, decide whether $c \equiv ab \mod (p-1)$. A bit $b \in \{0, 1\}$ is outputted, such that $b = 1$ if and only if $c \equiv ab \mod (p-1)$.

At first, one might assume that DDH is computationally infeasible, as in order to evaluate if $c = ab$, one would have to solve the discrete log problem to find a, b, and c. However, consider the following construct:

Definition 4 (Bilinear Map)**.** We call a function $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_3$ a bilinear map if for $a \in \mathbb{G}_1$, $b \in \mathbb{G}_2$, $x, y \in \mathbb{Z}$:

$$e(a^x, b^y) = e(a, b)^{xy}$$

Since a bilinear map is computationally efficient, it can be used to solve the DDH problem very easily by checking if the following statement is true:

$$e(g^a, g^b) = e(g, g^c)$$

as it is essentially checking if $e(g, g)^{ab} = e(g, g)^c$. It should be noted that the CDH problem remains hard even with bilinear maps, as bilinear maps cannot be used to generate the actual exponent.

Definitions 1-4 are based on [ZSS04].

We now define the BLS signature scheme as follows:

Definition 5 (BLS Signature Scheme)**.** The BLS signature scheme contains 3 protocols: key generation (K), signing (S), and verification (V) [BLS01]. More specifically:

- K: A prime p, a generator $g \in \mathbb{Z}_p$, and a secret key $sk \in \mathbb{Z}_p$ are chosen. There is also a public key $pk = g^{sk}$ (where exponentiation is done from within the residue $\mathbb{Z}_p$). pk, g, and p are publicized

- S: A message $m \in M$ is chosen from message space M. A signature $\sigma = m^{sk}$ is produced and publicized

- V: Given a bilinear map e, it is verified that $e(\sigma, g) = e(m, g^{sk})$. A bit $b \in \{0, 1\}$ is outputted, such that $b = 1$ if and only if $e(\sigma, g) = e(m, g^{sk})$

Due to DDH, verification is trivialized and easy to perform. By contrast, it is very difficult to forge a signature, as it is difficult to compute the private key sk due to CDH and the discrete logarithm problem. This provides a very robust signature scheme. Optionally, one could use a hash function on the message for additional security, and thus perform the same algorithm but use $H(m)$ as opposed to m, for some hash function $H(x)$.

Due to the use of exponents, key homomorphism is present in the scheme. For example, suppose one wished to split the private key pk between two participants and form pk_a and pk_b. It is possible to reconstruct the overall signature $\sigma = g^{pk}$ without either participant knowing their opposite private key. If we have a dealer give out pk_a and pk_b such that $pk = pk_a + pk_b$ and they are randomly distributed in the residue, then each participant can create their own signatures ($\sigma_a = g^{pk_a}$ and $\sigma_b = g^{pk_b}$ respectively). The participants can then compute $\sigma_a * \sigma_b = g^{pk_a} * g^{pk_b} = g^{(pk_a + pk_b)} = g^{pk} = \sigma$. Thus, we can see that there is a direct and easily computable relationship between the private key and the signature, as by manipulating simply the signatures, one can also manipulate the private key and therefore the overall signature. This property will be very useful later on in the paper.

3 Related Works

Firstly, we will establish preliminary notions that will be used to expand upon and better understand our monotonic signature scheme. We start with defining a signature scheme for a single party.

Definition 6 (Signature Scheme)**.** A signature Scheme S consists of 3 polynomial time algorithms (K, S, V):

- $K(k)$ This algorithm outputs sk (secret key) and pk (private key) with message space M using the security input k. sk is used for signing, and therefore should remain private for the party. pk is used for verification, and is thus publicized.

- $S(sk, m)$: Using the secret key sk and a message $m \in M$, a signature σ is generated and outputted
- $V(pk, m, \sigma)$: Using the public key pk, message m and the signature σ, a bit $b \in \{0, 1\}$ is outputted, where $b = 1$ denotes a valid signature

Digital signatures (both singular and multisignature algorithms) contain several important properties. One of these properties is *authentication*, as private ownership of the private key requires that any produced signature and verification of such signature could only have come from the specific party that has the private key. No other party could reconstruct or generate the private key, nor could they forge a signature under such a private key.

Another important property of digital signatures is *integrity*, in that the message that is signed may not be altered without invalidating the signature.

A final important property of digital signatures is *non-repudiation*, which is that once the signature has been signed, it is permanent and non-redactable. This is achieved through the publication of the public key, the signature, and the message. The combination of the three can allow any public participant to check the signature forever, where the message could only have been signed by the party having the private key.

We seek to more formally define notions of correctness and security for digital signature algorithms:

Definition 7 (Correctness)**.** A digital signature scheme S is correct, if for all security parameters $k \in N$, $K(k)$, sk, pk, and $m \in M$, we have that $\Pr[V(pk, m, S(sk, m)) = 1] = 1$.

Definition 8 (Security)**.** A signature scheme S is secure if for all adversaries A, for any message $m \in M$, and for any $K(k) \to (sk, pk)$, there exists a negligible function ϵ such that

$$\Pr[A(pk, m') \to \sigma', V(pk, m', \sigma') = 1] \leq \epsilon(k)$$

Definitions 6-8 are from [DS19].

Using these baseline signature definitions, we seek to move on to multisignature schemes. We first must formally define a method in which certain parties may join together in order to form a signature. We call such an object an access structure, where all combinations of valid participant subsets are contained within it. Let us now define an access structure, as it is a necessary construct in order to formally define a valid set of participants for a multisignature scheme:

Definition 9 (Access Structure)**.** We say that an access structure A of set U is a subset $A \subseteq 2^U$. Provided we have a set I such that $I \subseteq U$, if $I \in A$, we can define I as a qualified subset. [Bai+16]

Access structures are beneficial in order to formalize which combinations of parties may be used in order to successfully perform a multisignature scheme. Let us now define a Distributed Signature Scheme:

Definition 10 (Distributed Signature Scheme)**.** We define a signature scheme $S = (K, S, V)$. We denote a corresponding (A, n) distributed signature scheme (where n is the number of participants and A is an access structure) as a triple of polynomial time algorithms (DK, DS, DV). They are defined as follows:

- $DK(k)$: This algorithm takes in a security parameter k and outputs a grandmaster public and secret key, pk and sk respectively. Secret key shares $sk_1, sk_2, \ldots sk_n$ are distributed to parties $P_1, P_2, \ldots P_n$

- $DS(\{sk_i\}_{[i \in U]}, m)$: Given a subset of parties $U \in A$ and a message $m \in M$, they seek to reconstruct a signature $\sigma = m^{pk}$ using their secret key shares. The output is (σ, m). If $U \notin A$, the reconstruction process necessarily fails, and no valid signature is produced

- $DV(pk, m, \sigma)$: The protocol verifies the validity of the signature using the public key, and outputs a bit $b \in \{0, 1\}$, where $b = 1$ if and only if the signature is valid

The reconstruction process may be accomplished with a variety of methods, which only allows valid subsets as dictated by the access structure to successfully complete the protocol. Since the shares are randomized, it is impossible for an adversarial set of parties to reconstruct the signature without passing through the access structure.

Let us now formally define threshold signature scheme:

Definition 11 (Threshold Signature Scheme)**.** We define a signature scheme $S = (K, S, V)$. We denote a corresponding (t, n) threshold signature scheme (n is the number of parties, t is the threshold such that $1 \leq t \leq n$) as a triple of polynomial time algorithms (TK, TS, TV). They are defined as follows:

- $TK(k)$: This algorithm takes in a security parameter k and outputs a grandmaster public and secret key, pk and sk respectively. Secret key shares $sk_1, sk_2, \ldots sk_n$ are distributed to parties $P_1, P_2, \ldots P_n$.

- $TS(\{sk_i\}_{[i \in U]}, m)$: Given a subset of parties $|U| \geq t$ and a message $m \in M$, they seek to reconstruct a signature $\sigma = m^{pk}$ using their secret key shares. The output is (σ, m). If $|U| < t$, the reconstruction process necessarily fails, and no valid signature is produced

- $TV(pk, m, \sigma)$: The protocol verifies the validity of the signature using the public key, and outputs a bit $b \in \{0, 1\}$, where $b = 1$ if and only if the signature is valid [Bol03]

Some important properties for the threshold signature scheme are that the shares $sk_1, sk_2, \ldots sk_n$ are randomly distributed, and should have the same distribution as the baseline signature scheme key generation K. Additionally, typically it is the case that $TV = V$, as one simply uses the reconstruction protocol to create the signature that would otherwise have been generated by the baseline signature scheme S, and from there the baseline scheme can be used to verify the signature using V.

The threshold signature scheme may be trivially accomplished by using a secret sharing protocol with a threshold mechanism embedded. In such a way, each party gets a share s_i, and only if the number of parties exceeds t will they be able to reconstruct the grandmaster private key pk. Many such secret sharing schemes exist. Perhaps most famous one is Shamir Secret Sharing, which allows for threshold secret sharing by utilizing Lagrange interpolation. The shares are points on a polynomial, and the polynomial itself can be reconstructed if enough parties join together.

Let us also define the unate function:

Definition 12 (Unate function)**.** We say that a boolean function $f(x_1, x_2, \ldots x_n)$
(where $x_1, x_2, \ldots x_n \in \{0, 1\}$ and the function output is a bit $b \in \{0, 1\}$), is unate if for any x_i:

$$f(x_1, x_2 \ldots x_{i-1}, 1, x_{i+1}, \ldots x_n) \geq f(x_1, x_2 \ldots x_{i-1}, 0, x_{i+1}, \ldots x_n)$$

It is significant to note that all monotonic functions $f(x)$ can be described using a boolean circuit containing AND and OR gates, where AND and OR gates receive two binary inputs and output their corresponding functional values. However, it is assumed that one is able to replicate the outputs of the AND and OR gates into multiple future inputs, and we denote such replication as FANOUT (it receives 1 input and has multiple outputs that copy the input).

Let us now define a monotonic access structure:

Definition 13 (Monotonic Access Structure)**.** We say that an access structure A of set U is a monotone subset $A \subseteq 2^U$. Thus, for $\forall B, C$, if $B \in A$ and $B \subseteq C$, then $C \in A$. Provided we have a set I such that $I \subseteq U$, if $I \in A$, we can say that I is a qualified subset. We may form a bijective correspondence between sets of I that are in A in the following manner:

$$I \in A \iff f(I) = 1$$

where $f(x_1, x_2 \dots x_n)$ is a unate function as described earlier [Bai+16] [OSW07].

Monotone access structures are useful in that they allow for substantial variance and flexibility, as they simply require that the addition of more parties to a set attempting to pass the access structure will not decrease the chance of passing. This logic is also natural and useful for real world applications, as one could expect that the addition of participants/keys should naturally lead to an increased chance of success. The removing of participants/keys in order to have a higher chance, by contrast, is unnatural and not intuitive. Furthermore, such structures may be modeled with AND, OR, and FANOUT gates, due to their bijective correspondence with arbitrary unate functions. This grants much utility and practicality in modeling these access structures. Thus, we seek to form a multiparty signature scheme that has a monotonic access structure.

4 Protocol

Using the BLS signature scheme, we seek to craft a monotonic signature scheme that will produce a valid signature for any valid combination of valid input signatures. The general idea behind the scheme is to form a mechanism that is analogous to a garbled circuit: There will be input wires where valid signatures would be submitted, and there will be an output wire where a valid combined signature is produced.

As discussed earlier, a boolean monotonic function can be crafted through 2 gates (AND and OR). AND gates will require two valid signatures in order to propagate a valid signature, and OR gates will require only one valid signature in order to propagate a valid signature. We also use FANOUT, which accepts a valid signature, and will propagate multiple valid signatures.

The idea behind the mechanism is for there to exist a grandmaster public and private key. In order to generate the private keys for the individuals, the network propagates from the end node upwards. Once each party has their private key, they generate their signatures, and propagate through the circuit from the top to the bottom. We will denote the first phase as the *key generation phase* (as the private keys for each party will be created), the second phase as the *combined signature phase*, and the final phase as the *combined verification phase* (which simply verifies the combined signature with the grandmaster public key).

The above mechanism will use 3 gates: the AND, OR, and FANOUT gates that have already been stipulated. We will first describe the key generation phase through the bottom-up traversal of the wires, and then the combined signature phase will be described through the top-bottom traversal of the wires.

Let us now more formally define the notion of a circuit and introduce notation in order to more accurately describe the proposed signature scheme.

Definition 14 (Circuit)**.** A circuit $C : \{x_1, \dots x_n\} \to y$ is described by wires $\{w_1, ..., w_\ell\}$, where $\ell > n$, and gates $\{g_1, g_2, \dots g_\lambda\}$, where the number of gates is λ. Given a wire w_i, we call it an input wire if $i \in [1, n]$, an internal wire if $i \in [n+1, \ell - 1]$, and an output wire if $i = \ell$. Every internal wire connects two gates. However, the input wires are only connected to one gate, and so is the output wire. Furthermore, each gate is connected to 3 wires. As such, we have $\lambda = (2\ell + n + 1)/3$ gates.

Each wire $w_i, i \in [1, \ell]$ carries a value, whereas each gate $g_i, i \in [1, \lambda]$ is described by a tuple (w_a, w_b, w_c) of 3 wires and a type $t_i \in \mathsf{AND}, \mathsf{OR}, \mathsf{FANOUT}$. The type determines the characteristics of the tuple of 3 wires: if g_i has type $t_i \in \mathsf{AND}, \mathsf{OR}$, then g_i has two input wires $w_{i_{LI}}$ and $w_{i_{RI}}$ to the left and right respectively, and one output wire w_{i_O}. Finally, if $t_i = \mathsf{FANOUT}$, then g_i has two output wires $w_{i_{LO}}$ and $w_{i_{RO}}$, and one input wire w_{i_I}. WLOG the output wire w_{i_O} is not connected to a gate of type $t_\lambda = \mathsf{FANOUT}$.

There are two possible evaluations of a circuit: bottom-up and top-down. For a bottom-up evaluation of a circuit, a gate g_i is ready to be evaluated if and only if its output wires have been assigned. Thus, if $t_i = \mathsf{AND}, \mathsf{OR}$, the gate is ready to be evaluated if $w_{i_O} \neq \perp$. If $t_i = \mathsf{FANOUT}$, the gate is ready to be evaluated if $w_{i_{LO}} \neq \perp$ and $w_{i_{RO}} \neq \perp$. Evaluation of the circuit commences as follows:

- Given input y, assign $w_\ell = y$
- While $w_i = \perp$ for $\forall i \in [1, n]$, evaluate the first gate g_i that is *ready to be evaluated*:
  - If $t_i = \mathsf{AND}$, assign $w_{i_{LI}} = j$ and $w_{i_{RI}} = k$ such that $w_{i_O} = j \wedge k$.
  - If $t_i = \mathsf{OR}$, assign $w_{i_{LI}} = j$ and $w_{i_{RI}} = k$ such that $w_{i_O} = j \vee k$.
  - If $t_i = \mathsf{FANOUT}$, assign $w_{i_I} = j$, such that $j = w_{i_{LO}} = w_{i_{RO}}$ (Note that this does not necessarily mean the three numbers are *numerically equal*, for example in our signature scheme this means that all 3 private keys are valid, and therefore during the top-down part of the circuit evaluation, a valid input signature will lead to two valid output signatures)
- Output $(w_1, w_2, \dots w_n)$

For a top-down evaluation of a circuit, a gate g_i is ready to be evaluated if and only if its input wires have been assigned. Thus, if $t_i = \mathsf{AND}$, the gate is ready to be evaluated if $w_{i_{LI}} \neq \perp$ and $w_{i_{RI}} \neq \perp$. If $t_i = \mathsf{OR}$, the gate is ready to be evaluated if $w_{i_{LI}} \neq \perp$ or $w_{i_{RI}} \neq \perp$. If $t_i = \mathsf{FANOUT}$, the gate is ready to be evaluated if $w_{i_I} \neq \perp$. Evaluation of the circuit commences as follows:

- 1. Parse the input $x_1, x_2, \dots x_n$, and assign $w_i = x_i$ for all $i \in [1, n]$
- 2. While $w_\ell = \perp$, find the first gate g_i which is *ready to be evaluated*:
  - If $\mathrm{t}_i = \mathsf{AND}$, then assign $w_{i_O} = w_{i_{LI}} \wedge w_{i_{RI}}$
  - If $\mathrm{t}_i = \mathsf{OR}$, then assign $w_{i_O} = w_{i_{LI}} \vee w_{i_{RI}}$
  - If $\mathrm{t}_i = \mathsf{FANOUT}$, then assign $w_{i_{LO}} = w_I$ and $w_{i_{RO}} = w_I$
- Output w_ℓ. [Bai+16]

The above definition can easily be classified as a binary circuit should all the inputs, outputs, wires, and gates accept binary inputs. As was discussed earlier, all unate functions can be represented through AND and OR gates. Therefore, unate functions may be represented through a circuit. In our monotonic signature scheme, we seek to utilize a similar concept, but instead of evaluating the circuit with bits, it would be evaluated with valid/invalid signatures (a valid signature can be thought of as a 1, and an invalid signature as a 0). In this model, the gates perform operations on these signatures and output a valid or invalid signature based on the validity of the input signatures (as described earlier).

Thus, for our monotone signature scheme, we will have a key generation phase, a combined signature phase, and a combined verification phase. In the key generation phase, the grandmaster private key has to be distributed to the participants so that they can later reconstruct the valid signature. Thus, the algorithm will traverse up the circuit starting from the bottom, and assign wires based on lower wires and the logic gates. When all of the input wires have been assigned a private key, they will be distributed to each party. For the combined signature phase, a valid set of parties as dictated by the access structure will first individually craft their own signatures, and then traverse down the circuit, while following the logic of each gate accordingly. A final signature will be outputted at the very end through the output wire. Finally, the signature may be verified with the grandmaster public key in the combined verification phase. Let us describe in detail each of the 3 phases:

Key Generation Phase: Using the BLS scheme, we create a *grandmaster public key pk* and a *grandmaster secret key sk*. Given a unate function $f(x)$, we create a circuit C such that it models the function. Using C, we assign $w_\ell = sk$. From there, we perform a bottom-up traversal of the circuit, and we evaluate a gate (and therefore assign private keys to the input wires of the gate) if the output wire has been assigned a private key. Thus, for each gate g_i that is ready to be evaluated:

- If $t_i = \mathsf{AND}$: Given an output wire $w_{i_O} = sk_{i_O}$ and its corresponding private key, we assign $w_{i_{LI}} = sk_{i_{LI}}$ and $w_{i_{RI}} = sk_{i_{RI}}$ such that $sk_{i_{LI}} + sk_{i_{RI}} = sk_{i_O}$.
- If $t_i = \mathsf{OR}$: Given an output wire $w_{i_O} = sk_{i_O}$ and its corresponding private key, we assign $w_{i_{LI}} = sk_{i_O}$ and $w_{i_{RI}} = sk_{i_O}$.
- If $t_i = \mathsf{FANOUT}$: given two output wires $w_{i_{LO}} = sk_{i_{LO}}$ and $w_{i_{RO}} = sk_{i_{RO}}$, we sample a random integer $sk_{i_I} \in \mathbb{Z}_p$ and assign $w_{i_I} = sk_{i_I}$. Two variables $P_{i_L} = sk_{i_{LO}} * sk_{i_I}^{-1}$, $P_{i_R} = sk_{i_{RO}} * sk_{i_I}^{-1}$, which are stored and publicized

After each gate has been evaluated, the n input wires $w_1, \ldots w_n$ are assigned to each party $P_1, \ldots P_n$ through private means of communication (e.g. a private, secure channel) such that each party respectively has $sk_1, \ldots sk_n$. Thus, each party has their own private keys.

Combined Signature Phase: Each party $P_1, \ldots P_n$ computes their own signature $\sigma_1, \ldots \sigma_n$, where $\sigma_i = m^{sk_i}$. The wires in the circuit are reset and for each input wire we assign $w_i = \sigma_i$, where $i \in [1, \ell]$. Afterwards, a top-down traversal of the circuit is performed, each gate is evaluated (and therefore signatures are assigned to the output wires of the gate) if the input wires have been assigned a signature. Thus, for each gate g_i that is ready to be evaluated:

- If $t_i = \mathsf{AND}$: Given input wires $w_{i_{LI}} = \sigma_{i_{LI}}$ and $w_{i_{RI}} = \sigma_{i_{RI}}$, we assign $w_{i_O} = \sigma_{i_{LI}} * \sigma_{i_{RI}} = m^{sk_{i_{LI}}} * m^{sk_{i_{RI}}} = m^{(sk_{i_{LI}} + sk_{i_{RI}})} = m^{sk_{i_O}} = \sigma_{i_O}$
- If $t_i = \mathsf{OR}$: Given input wires $w_{i_{LI}} = \sigma_{i_{LI}}$ and $w_{i_{RI}} = \sigma_{i_{RI}}$, choose either signature and assign to w_{i_O}, as $\sigma_{i_{LI}} = \sigma_{i_{RI}} = \sigma_{i_O}$ due to $sk_{i_{LI}} = sk_{i_{RI}} = sk_{i_O}$
- If $t_i = \mathsf{FANOUT}$: Given an input wire $w_{i_I} = \sigma_{i_I}$, assign $w_{i_{LO}} = \sigma_{i_I}^{P_{i_L}} = (m^{sk_{i_I}})^{(sk_{i_{LO}} * sk_{i_I}^{-1})} = m^{(sk_{i_I} * sk_{i_{LO}} * sk_{i_I}^{-1})} = m^{sk_{i_{LO}}} = \sigma_{i_{LO}}$ and $w_{i_{RO}} = \sigma_{i_I}^{P_{i_R}} = (m^{sk_{i_I}})^{(sk_{i_{RO}} * sk_{i_I}^{-1})} = m^{(sk_{i_I} * sk_{i_{RO}} * sk_{i_I}^{-1})} = m^{sk_{i_{RO}}} = \sigma_{i_{RO}}$

After all the gates are evaluated, the final wire w_ℓ is publicized, as $w_\ell = \sigma$ is the final combined signature.

Combined Verification Phase: The final signature σ is verified with the grandmaster public key using a bilinear map, as is per usual with the BLS scheme.

Now that we have precisely described each of the 3 phases of the algorithm, let us verify the correctness of the algorithm by verifying that each gate performs its task correctly. For an AND gate, it will only output a valid signature if and only if it received two valid signatures from the input wires. This is in fact true: WLOG if $w_{i_{LI}} = \sigma'_{i_{LI}}$ for a gate g_i, then $w_{i_O} = \sigma'_{i_{LI}} * \sigma_{i_{RI}} = m^{sk'_{i_{LI}}} * m^{sk_{i_{RI}}} = m^{(sk'_{i_{LI}} + sk_{i_{RI}})} \neq \sigma_{i_O}$. The OR gates require only one valid signature, due to duplication of the private key during the key generation phase. By using variables that are publicly available, FANOUT gates are able to compute two valid signatures from one, as is described in the scheme.

Since the correctness has been verified, let us also verify the privacy of the scheme. The AND gates are secure, as WLOG an adversary A could not use their private key $sk_{i_{LI}}$ to compute the other private key $sk_{i_{RI}}$, as the adversary may compute $m^{sk_{i_O}}/m^{sk_{i_{LI}}} = m^{sk_{i_{RI}}}$, but they may not compute $sk_{i_{LI}}$ due to the difficulty of the discrete logarithm problem. The security of an OR gate is not necessary, however, since if the adversary already has a valid key, they do not need the other wire's key (which is already identical) in order to pass the OR gate. However, if they do not have the private key for the OR gate, the adversary cannot forge a signature. Finally, the FANOUT gate is secure, as WLOG an adversary cannot use $P_{i_L} = sk_{i_{LO}} * sk_{i_I}^{-1}$ to compute $sk_{i_{LO}}$ without possessing sk_{i_I}. Thus, none of the gates could be forced to artificially create a valid signature given an adversary.

It should be noted, however, that the parties must be semi-honest, and not malicious. This is especially obvious in the OR gate, as the assumption is that there is no adversary A such that they deliberately corrupt a signature, use it in one of the input wires for an OR gate, and therefore deliberately cause the OR gate to report false.

We now seek to prove correctness and privacy of the scheme. Correctness can be proven by inspection, and the soundness of the scheme (the inability of a set of parties not in the access structure to generate a valid signature) is also self evident from BLS. We seek to prove privacy by generating a simulator that can first be used to show the privacy of the scheme if the circuit has no FANOUT gates. We will then use the same simulator in order to show the privacy with the existence of FANOUT gates by showing that if the circuit is private with one less FANOUT gate, then it is private with its current number of FANOUT gates.

Definition 15 (Privacy)**.** By observing the recombined signature, any computationally efficient adversary learns nothing about message m.

Note that this is a weak notion of privacy, as realistically all signatures used during the recombination phase might be public. However, the proof shown can be similarly adapted to intermediary signatures, as such signatures can be thought of as sub-computations on a smaller part of the overall circuit.

Proof of Privacy: *The monotonic signature scheme is private under the assumption that the BLS signature scheme is private.* [BLS01]

We prove the privacy with a simulator. In the high level, we wish to create a simulator that does not know message m but generates a signature σ' that has the exact same distribution as when generated with the actual signature scheme. In this way, we show that the adversary cannot distinguish between the two values, and therefore the recombined signature leaks no information regarding message m.

Given a monotonic signature scheme S with circuit C, a monotone access structure A, a message m, and an adversary, we can always construct a simulator Sim:

1. Sim does the setup with a randomized grand master public and secret key (pk' and sk') respectively.

2. Sim executes the key generation phase, and generates n secret keys $sk'_1, \ldots sk'_n$ for hypothetical parties $P'_1, \ldots P'_n$ (we do not have real parties, but Sim runs them virtually).

3. Sim randomly samples partial signatures for all the parties from the signature space (without a message), if the secret keys for Party i and j are the same, Sim only samples a single signature and assigns it for i and j.

4. Sim executes the combined signature phase using the partial signatures, which generates a signature $w_\ell = \sigma'$.

We now prove that the distribution of σ' (the randomized recombined signature value generated from Sim) is computationally indistinguishable from the distribution of $w_\ell = \sigma$ (the actual recombined signature from the signature scheme S).

First, we note that $\sigma = w_\ell$ is the real view, as seen by final computation performed through the signature scheme S, whereas σ' is a random value seen in the simulated view. We also note the privacy of BLS, namely that given a BLS scheme, the signature $\sigma = m^{sk}$ gives no information pertaining to message m should the message not be released, as the signature has the same distribution as the key used. [BLS01]. We now first consider circuits without FANOUT gates, and afterwards we consider circuits with FANOUT gates.

Without FANOUT Gates One may observe that if a circuit C contains no FANOUT gates, the output signature is simply $w\ell = \prod_{i \in \mathsf{Set}} \sigma_i$, where $\sigma_i = m^{sk_i}$ for each party P_i and Set is a subset of the input parties. This is necessarily the case since an OR gate simply outputs one of two input wires (which are identical) and an AND gate simply outputs the product of two input wires. Thus, $w\ell = \prod_{i \in \mathsf{Set}} \sigma_i = m^{\sum_{i \in \mathsf{Set}} sk_i}$, and since the sum of random values is a random value, what is essentially computed is m^r for some random r.

If we now consider Sim, using the same logic the output signature is simply $w\ell = \prod_{i \in \mathsf{Set}} r_i$ where r_i is a random value. The product of random values is a random value, and thus Sim generates a random value. Finally, we note that under the BLS assumption, the signatures generated from S have the same random distribution as the keys used. Therefore, the distribution of the reconstructed signatures from S and Sim have the same distribution.

With FANOUT Gates We now seek to use an induction-like method to remove FANOUT gates, and ultimately decompose the circuit C into other circuits without FANOUT gates. We will first create a series of circuits $(C^*_0, C^*_1, \ldots C^*_f)$. We first define C^*_f as a circuit with an input length of $n^*_f = n + 2f$ wires (it receives the original input wires from C and two wires for each FANOUT gate, denoted with f) and contains the wires and gates that can reach w_ℓ without encountering a FANOUT gate. We then find the first input wire that is connected to a FANOUT gate during the previous search, and define C^*_{f-1} as a circuit with input length $n^*_{f-1} = n + 2(f-1)$ that contains all the wires and gates to reach the aforementioned input wire without encountering a FANOUT gate. We define $C^*_{f-2}, \ldots C^*_0$ similarly, with C^*_0 being the final definition of such circuits and having a maximum of $n_0^* = n$ input wires. We note that none of the inputs of C_0^* come from a FANOUT gate. A key observation is that the above circuits do not contain FANOUT gates.

Given that the original circuit C accepts n inputs $x_1, x_2, \dots x_n$ (where $x_i \in \{0,1\}$), we now generate more inputs as follows: $x_{n+1} = x_{n+2} = C_0^*(x_1, x_2, \dots x_n)$, $x_{n+3} = x_{n+4} = C_1^*(x_1, x_2, \dots x_n)$, and we end with $x_{n_f^*-1} = x_{n_f^*} = C_f^*(x_1, x_2, \dots x_n)$. Due to the fact that the new added inputs represent FANOUT replication (as each circuit leads to a FANOUT gate, which is then duplicated into two variables), we may write:

$$C(x_1, x_2, \dots x_n) = C_f^*(x_1, x_2, \dots x_{n_f^*})$$

The above statement is true for any $x_1, x_2, \dots x_{n_f^*} \in \{0,1\}$. We also construct circuits $(C_0^{'}, C_1^{'}, \dots C_f^{'})$ which do contain FANOUT gates, albeit less than the original circuit C. A circuit $C_i^{'}$ has $n_i^{'} = n + 2i$ input wires and $f - i$ FANOUT gates. We set $C_f^{'} = C_f^*$ and

$$C_{i-1}^{'} = C_i^{'}(x_1, \dots x_{n_{i-1}^{'}}, C_i^*(x_1, \dots x_{n_{i-1}^{'}}), C_i^*(x_1, \dots x_{n_{i-1}^{'}}))$$

We can clearly see that $C = C_0^{'}$. We now construct a series of adversaries such that each adversary D_i receives respectively: $\sigma_0, \sigma_1, \dots \sigma_{n_f^{'}}$, where the inputs are generated from a monotonic signature scheme with circuit $C_i^{'}$ with randomized signatures. We have already shown that a monotonic signature scheme is private given that it does not contain FANOUT gates. Hence, since D_f receives circuit $C_f^{'} = C_f^*$ which does not contain FANOUT gates, the output wire is private. We will now argue inductively that if adversary D_{i-1} successfully breaks privacy by outputting the true value, then so can D_i. This will form a contradiction, as by finite descent the implication is that the privacy of D_f will be compromised. Thus, we can conclude that adversary D_0 can only compromise the security of the scheme should the adversary compromise the BLS and discrete logarithm problem assumptions.

For the induction part of the proof, we need to show that adversary D_i may construct a signature scheme for adversary D_{i-1} (as adversary D_{i-1} has an additional FANOUT gate). By creating this connection, if adversary D_{i-1} compromises the privacy of the signature scheme with circuit $C_{i-1}^{'}$, then so may D_i with circuit $C_i^{'}$. A simple consideration is to add a FANOUT gate, and re-run the key generation phase for circuit C_i^* with the added gate. However, a difficulty arises when we consider that the signatures $\sigma_0^*, \sigma_1^*, \dots \sigma_{n+2i-2}^*$ (the valid/invalid signature corresponding with inputs $x_0^*, x_1^*, \dots x_{n+2i-2}^*$) for circuit C_i should correspond with all other existing signatures between circuits. In particular, $\sigma_j^* = \sigma_j^{'}$ for $0 \leq j \leq n + 2j$. We thus consider 3 cases:

1. **C_i^* outputs 1:** In this case, no values from the FANOUT gate need to remain hidden from adversary D_{i-1}, thus D_i runs the signature generation protocol using circuit C_i^*, computes some product of signatures σ_t. Using the public variables P_{i_L} and P_{i_R}, $w_{i_{LO}} = \sigma_t^{P_{i_L}}$ and $w_{i_{RO}} = \sigma_t^{P_{i_R}}$ are computed, then $\sigma_0^* = \sigma_0^{'} || (w_{i_{LO}}, w_{i_{RO}})$ and $\sigma_j^* = \sigma_j^{'}$ is true for remaining j. This is given as input for D_{j-1}.

2. **C_i outputs 0, however both $\sigma_{n+2i-1}^{'}$ and $\sigma_{n+2i}^{'}$ are known:** One case where this may occur is if the outputs of the FANOUT gate lead to the inputs of two OR gates. In this case, the signatures are already known through the computation of the other two legs of the OR gate, as we have defined OR gates to have identical inputs (both equal to the output). In this case, D_j can simply encrypt the two signatures $\sigma_{n+2i-1}^{'}$ and $\sigma_{n+2i}^{'}$ randomly. Since $C_i^* = 0$ (i.e. the circuit fails) and it does not contain FANOUT gates, the adversary D_{i-1} cannot tell the difference between different random encryptions without compromising the BLS and discrete logarithm assumptions.

3. C_i **outputs 0, however either** $\sigma^{'}_{n+2i-1}$ **or** $\sigma^{'}_{n+2i}$ **(or both) are not known:** WLOG let D_{i-1} know $\sigma^{'}_{n+2i-1}$ and not $\sigma^{'}_{n+2i}$. With σ_t as defined earlier, D_{i-1} then receives inputs $w_{i_{LO}} = \sigma_t^{\sigma^{'}_{n+2i-1}}$ and $w_{i_{RO}} = \sigma_t^r$ for a random r. If D_i outputs the value of the output wire, then D_{i-1} can compute r and compromise the privacy of the BLS security scheme, thus forming a contradiction. Similar reasoning may be used when both $\sigma^{'}_{n+2i-1}$ and $\sigma^{'}_{n+2i}$ are not known.

[Bai+16]

5 Limitation

Note that during the setup phase, for every FANOUT gate we publish two public values $P_{i_L} = sk_{i_{LO}} * sk_{i_I}^{-1}$ and $P_{i_R} = sk_{i_{RO}} * sk_{i_I}^{-1}$. However, an adversary could easily compute $P_{i_L} * P_{i_R}^{-1} = sk_{i_{LO}} * sk_{i_{RO}}^{-1}$, and therefore leak information through the product of two secret keys. Such an issue poses limitations to the extent of the security of the scheme, although there are methods to limit this issue, including key-length doubling (and therefore one must also limit the maximum depth of FANOUT gates to prevent exponential growth), or the use of additional secret encryption keys.

6 Applications

The signature scheme may be used for any application requiring a joint signature scheme that utilizes a monotonic function. One such application is for a joint verification of a message, such as a letter, email, document, etc. Previously, such signing of messages was done individually, and a joint signature scheme would be simple in that it either just requires the parties to pass an unweighted threshold or for the parties to pass a threshold where each party has relatively simple weights (no irrational or complex rational numbers). This could be useful in more basic applications, but consider the following example: suppose you have 1 president and 5 subordinates, and one wishes to craft a scheme such that if the president signs the document or at least 3 of the 5 subordinates sign the document. Although this could still be solved with a weighted threshold signature scheme, it could be very easy to imagine how additional constrains and more complex functions could make weighted threshold schemes infeasible. By contrast, with the monotonic signature scheme, complex functions can be modeled very easily, as a binary circuit can model any monotonic function, and therefore this signature scheme is useful for complex functions such as the above example.

Another potential use of the signature scheme is for hierarchical access control, where the monotonic signature scheme is used to authenticate users. For example, consider an office building, where you have 1 president and 5 subordinates, and in order to enter the building you need at least the president or 3 of the 5 subordinates. This could be solved by requiring the users to sign a message, and authenticate them into the building only if the signature is valid. Thus, the monotonic signature scheme may be used again, in this situation to authenticate users and allow for hierarchical access control.

7 Conclusion

To conclude, a new signature scheme for multiple parties was created that utilized a circuit and the BLS signature scheme, and it has monotonic properties. This could have real, practical applications, as are described in the previous section. There are several areas for additional research.

One area for additional research is to look into different signature schemes with key homomorphism, particularly schemes that are randomized. Such schemes provide an additional layer of security, and thus allow for a more robust scheme.

Another area for future research is for the use of the protocol to solve existing problems. In particular, Dolev-Strong, a famous result to solve the broadcast problem, currently uses a signature protocol. Its efficiency could be further improved with the monotonic signature scheme, as opposed to using a signature scheme for each individual.

Finally, another area for development is a way to reduce the memory required for FANOUT gates. Currently, two variables need to be created and stored for FANOUT gates, which is inefficient due to the potential for a large number of FANOUT gates. A reduction in the number of variables necessary for FANOUT could lead to a dramatic decrease in memory usage.

References

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

[Bol03] Alexandra Boldyreva. "Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme". In: *International Workshop on Public Key Cryptography*. Springer. 2003, pp. 31–46.

[ZSS04] Fangguo Zhang, Reihaneh Safavi-Naini, and Willy Susilo. "An efficient signature scheme from bilinear pairings and its applications". In: *International Workshop on Public Key Cryptography*. Springer. 2004, pp. 277–290.

[OSW07] Rafail Ostrovsky, Amit Sahai, and Brent Waters. "Attribute-based encryption with non-monotonic access structures". In: *Proceedings of the 14th ACM conference on Computer and communications security*. 2007, pp. 195–203.

[Bai+16] Ge Bai et al. "Non-interactive verifiable secret sharing for monotone circuits". In: *International Conference on Cryptology in Africa*. Springer. 2016, pp. 225–244.

[DS19] David Derler and Daniel Slamanig. "Key-homomorphic signatures: definitions and applications to multiparty signatures and non-interactive zero-knowledge". In: *Designs, Codes and Cryptography* 87.6 (2019), pp. 1373–1413.

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