Threshold Signatures: Efficient Constructions and Applications in Blockchains

Zhuo Cai

Department of Computer Science, HKUST

April 30, 2024

Table of Contents

Background: Digital Signatures

In cryptocurrencies, everyone has a pair of public/secret keys.
Alice has pk_A and sk_A. Others know pk_A, but only Alice knows sk_A.

Examples

Alice wants to pay 1 coin to Bob and propagates a message "Alice pays 1 coin to Bob".
Can Alice spend Bob's coin by broadcasting "Bob pays 2 coins to Alice"?

Background: Digital Signatures

In cryptocurrencies, everyone has a pair of public/secret keys.
Alice has pk_A and sk_A. Others know pk_A, but only Alice knows sk_A.

Examples

Alice wants to pay 1 coin to Bob and propagates a message "Alice pays 1 coin to Bob".
Can Alice spend Bob's coin by broadcasting "Bob pays 2 coins to Alice"?

Alice should not be able to add the message to blockchain.

Background: Digital Signatures

In cryptocurrencies, everyone has a pair of public/secret keys.
Alice has pk_A and sk_A. Others know pk_A, but only Alice knows sk_A.

Examples

Alice wants to pay 1 coin to Bob and propagates a message "Alice pays 1 coin to Bob".
Can Alice spend Bob's coin by broadcasting "Bob pays 2 coins to Alice"?

Alice should not be able to add the message to blockchain.

How can we prevent Alice from spending Bob's coins?

Background: Digital Signatures

In cryptocurrencies, everyone has a pair of public/secret keys.
Alice has pk_A and sk_A. Others know pk_A, but only Alice knows sk_A.

Examples

Alice wants to pay 1 coin to Bob and propagates a message "Alice pays 1 coin to Bob".
Can Alice spend Bob's coin by broadcasting "Bob pays 2 coins to Alice"?

Alice should not be able to add the message to blockchain.

How can we prevent Alice from spending Bob's coins?
- Alice should sign on her message: a digital signature σ_A generated using secret key sk_A, and can be verified using pk_A.

Background: Digital Signatures

In cryptocurrencies, everyone has a pair of public/secret keys.
Alice has pk_A and sk_A. Others know pk_A, but only Alice knows sk_A.

Examples

Alice wants to pay 1 coin to Bob and propagates a message "Alice pays 1 coin to Bob".
Can Alice spend Bob's coin by broadcasting "Bob pays 2 coins to Alice"?

Alice should not be able to add the message to blockchain.

How can we prevent Alice from spending Bob's coins?
- Alice should sign on her message: a digital signature σ_A generated using secret key sk_A, and can be verified using pk_A.
- Alice cannot forge Bob's signature to spend Bob's coins.

Multi Signatures

Alice and Bob jointly own a coin c. Only if Alice and Bob both agree, the coin c can be spent.

Multi Signatures

Alice and Bob jointly own a coin c. Only if Alice and Bob both agree, the coin c can be spent.
Solution:

- Alice signs on $m =$"A&B transfer coin c to Carol": $\sigma_A(m)$.
- Bob also signs on $m =$"A&B transfer coin c to Carol": $\sigma_B(m)$.
- (Verification): The miner includes m only if he sees both $\sigma_A(m)$ and $\sigma_B(m)$.

Multi Signatures

For a group of n members, $\{P_1, P_2, \ldots, P_n\}$, a signature is valid if only if all n members all agree to generate it.

Threshold Signatures

Alice, Bob and Carol are committee members of company that controls many coins.
Any two of them can spend these coins.

Threshold Signatures

Alice, Bob and Carol are committee members of company that controls many coins. Any two of them can spend these coins.
Solution: (When Alice and Carol agree to donate a coin to HKUST)

- Alice and Carol each signs on m: $\sigma_A(m), \sigma_C(m)$.
- (Verification): the miner includes m if he sees two of $\{\sigma_A(m), \sigma_B(m), \sigma_C(m)\}$.

Threshold Signatures

For a group of n members, $\{P_1, P_2, \ldots, P_n\}$, a signature is valid if only if a subset of *at least* t members agree to generate it.

Efficiency

Is the previous naïve threshold signature efficient? ($t = \Theta(n)$ e.g. $t = (n-1)/2$)

- Signature generation time: $\Theta(n)$.
- Signature size: $\Theta(n)$.
- Verification time: $\Theta(n)$.

Efficiency

Is the previous naïve threshold signature efficient? ($t = \Theta(n)$ e.g. $t = (n-1)/2$)

- Signature generation time: $\Theta(n)$.
- Signature size: $\Theta(n)$.
- Verification time: $\Theta(n)$.

Settings:

- A signature is verified many times (all blockchain nodes have to verify each transaction).
  - Try reducing the signature size/verification time, even at the cost of slow generation time.

Efficiency

Is the previous naïve threshold signature efficient? ($t = \Theta(n)$ e.g. $t = (n-1)/2$)

- Signature generation time: $\Theta(n)$.
- Signature size: $\Theta(n)$.
- Verification time: $\Theta(n)$.

Settings:

- A signature is verified many times (all blockchain nodes have to verify each transaction).
  - Try reducing the signature size/verification time, even at the cost of slow generation time.
- The group might generate many signatures!
  - Try reducing the cost of each signature (Gen+Ver), even at the cost of some expensive setup/precomputation.

Table of Contents

Component 1: digital signature scheme

Digital Signature Scheme: $\mathsf{SGN} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$

- $\mathsf{Setup}(1^\lambda) \rightarrow \mathsf{par}$ (probabilistic)
- $\mathsf{Gen}(\mathsf{par}) \rightarrow (\mathsf{pk}, \mathsf{sk})$ (probabilistic)
- $\mathsf{Sig}(\mathsf{sk}, \mathsf{m}) \rightarrow \sigma$ (probabilistic)
- $\mathsf{Ver}(\mathsf{pk}, \mathsf{m}, \sigma) \rightarrow \mathsf{b}$ (deterministic)

Correctness: $Pr[(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(\mathsf{par}), \mathsf{Ver}(\mathsf{pk}, \mathsf{m}, \mathsf{Sig}(\mathsf{sk}, \mathsf{m}))] = 1$.
EUF-CMA Security (Existentially Unforgeable under Chosen Message Attack): any probabilistic polynomial time algorithm ($poly(\lambda)$ time) can forge a signature for a new message with only negligible probability $o(1/poly(\lambda))$.

Examples: RSA, ElGamal, Schnorr, **BLS**, EdDSA.

Discrete log Problem

Many cryptography protocols are based on the hardness of the discrete log problem.

Using a cyclic multiplicative group $\mathbb{G}$, with a generator g of prime order q,
$\mathbb{G} = \{1, g, g^2, g^3, \ldots, g^{q-1}\}$.
$\mathbb{G}$ can be a subgroup of a multiplicative group $\mathbb{Z}_p^\times$, or elliptic curves.

[1] Alessandro Amadori, Federico Pintore, and Massimiliano Sala. "On the discrete logarithm problem for prime-field elliptic curves". In: *Finite Fields Their Appl.* 51 (2018), pp. 168–182.

Discrete log Problem

Many cryptography protocols are based on the hardness of the discrete log problem.

Using a cyclic multiplicative group $\mathbb{G}$, with a generator g of prime order q,
$\mathbb{G} = \{1, g, g^2, g^3, \ldots, g^{q-1}\}$.
$\mathbb{G}$ can be a subgroup of a multiplicative group $\mathbb{Z}_p^\times$, or elliptic curves.

Let $\lambda = \log q$.
- Given any input $x \in \mathbb{Z}_p = \{0, 1, \ldots, q-1\}$, it's efficient ($poly(\lambda)$ time) to compute $y = g^x \in \mathbb{G}$.

[1] Alessandro Amadori, Federico Pintore, and Massimiliano Sala. "On the discrete logarithm problem for prime-field elliptic curves". In: *Finite Fields Their Appl.* 51 (2018), pp. 168–182.

Discrete log Problem

Many cryptography protocols are based on the hardness of the discrete log problem.

Using a cyclic multiplicative group $\mathbb{G}$, with a generator g of prime order q,
$\mathbb{G} = \{1, g, g^2, g^3, \ldots, g^{q-1}\}$.
$\mathbb{G}$ can be a subgroup of a multiplicative group $\mathbb{Z}_p^\times$, or elliptic curves.

Let $\lambda = \log q$.
- Given any input $x \in \mathbb{Z}_p = \{0, 1, \ldots, q-1\}$, it's efficient ($poly(\lambda)$ time) to compute $y = g^x \in \mathbb{G}$.

Discrete log Problem (DL)

Given a random $y \in \mathbb{G}$, output $x \in \mathbb{Z}_q$ such that $g^x = y$.

Best known algorithms for DL in arbitrary elliptic curve groups of size q bits take $O(\sqrt{q}) = O(\exp(\lambda/2))$ time[1].

[1] Alessandro Amadori, Federico Pintore, and Massimiliano Sala. "On the discrete logarithm problem for prime-field elliptic curves". In: *Finite Fields Their Appl.* 51 (2018), pp. 168–182.

Diffie-Hellman Problem

Computational Diffie-Hellman Problem (CDH)

Given (g, u, v), three random elements of $\mathbb{G}$, to compute $h = g^{\log_g u \log_g v}$.

Diffie-Hellman Problem

Computational Diffie-Hellman Problem (CDH)

Given (g, u, v), three random elements of $\mathbb{G}$, to compute $h = g^{\log_g u \log_g v}$.

Difficulty of CDH $\leq$ difficulty of DL. If DL is easy, then $\log_g u$ and $\log_g v$ can be computed efficiently.

Diffie-Hellman Problem

Computational Diffie-Hellman Problem (CDH)

Given (g, u, v), three random elements of $\mathbb{G}$, to compute $h = g^{\log_g u \log_g v}$.

Difficulty of CDH $\leq$ difficulty of DL. If DL is easy, then $\log_g u$ and $\log_g v$ can be computed efficiently.

Decisional Diffie-Hellman Problem (DDH)

Given (g, u, v, h), four elements of $\mathbb{G}$, which with equal probability can be either all random elements of $\mathbb{G}$ or have the property that $\log_g u = \log_v h$, to output 0 in the former case and 1 in the latter case.

Diffie-Hellman Problem

Computational Diffie-Hellman Problem (CDH)

Given (g, u, v), three random elements of $\mathbb{G}$, to compute $h = g^{\log_g u \log_g v}$.

Difficulty of CDH $\leq$ difficulty of DL. If DL is easy, then $\log_g u$ and $\log_g v$ can be computed efficiently.

Decisional Diffie-Hellman Problem (DDH)

Given (g, u, v, h), four elements of $\mathbb{G}$, which with equal probability can be either all random elements of $\mathbb{G}$ or have the property that $\log_g u = \log_v h$, to output 0 in the former case and 1 in the latter case.

Difficulty of DDH $\leq$ difficulty of CDH. If CDH is easy, then computing $g^{\log_g u \log_g v}$ and comparing with h is an efficient algorithm for DDH.

Gap-DH and Bilinear Pairing [BF01][2]

Gap-DH groups: for some groups $\mathbb{G}$, the *Computational* DH problem is hard but the *Decisional* DH problem is easy.

[2] Dan Boneh and Matthew K. Franklin. "Identity-Based Encryption from the Weil Pairing". In: *CRYPTO*. vol. 2139. Lecture Notes in Computer Science. Springer, 2001, pp. 213–229.

Gap-DH and Bilinear Pairing [BF01][2]

Gap-DH groups: for some groups $\mathbb{G}$, the *Computational* DH problem is hard but the *Decisional* DH problem is easy.
Works on Weil pairing showed the existence of such GDH groups.

[2] Dan Boneh and Matthew K. Franklin. "Identity-Based Encryption from the Weil Pairing". In: *CRYPTO*. vol. 2139. Lecture Notes in Computer Science. Springer, 2001, pp. 213–229.

Gap-DH and Bilinear Pairing [BF01][2]

Gap-DH groups: for some groups $\mathbb{G}$, the *Computational* DH problem is hard but the *Decisional* DH problem is easy.
Works on Weil pairing showed the existence of such GDH groups.

Bilinear Pairing

A mapping $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a bilinear map for groups $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ with generators g_1, g_2, g_T of prime order q, if it is:

- bilinear: $e(g_1^x, g_2^y) = e(g_1, g_2)^{xy}$.
- non-degenerate: $e(g_1, g_2) \neq 1_{\mathbb{G}_T}$.
- efficient: there is an efficient algorithm to compute $e(g_1^x, g_2^y)$.

[2] Dan Boneh and Matthew K. Franklin. "Identity-Based Encryption from the Weil Pairing". In: *CRYPTO*. vol. 2139. Lecture Notes in Computer Science. Springer, 2001, pp. 213–229.

Gap-DH and Bilinear Pairing [BF01][2]

Gap-DH groups: for some groups $\mathbb{G}$, the *Computational* DH problem is hard but the *Decisional* DH problem is easy.
Works on Weil pairing showed the existence of such GDH groups.

Bilinear Pairing

A mapping $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a bilinear map for groups $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ with generators g_1, g_2, g_T of prime order q, if it is:

- bilinear: $e(g_1^x, g_2^y) = e(g_1, g_2)^{xy}$.
- non-degenerate: $e(g_1, g_2) \neq 1_{\mathbb{G}_T}$.
- efficient: there is an efficient algorithm to compute $e(g_1^x, g_2^y)$.

With a bilinear pairing from $\mathbb{G} \times \mathbb{G}$, a DDH problem for $(g, g^x, H(m), \sigma)$ is efficiently solved by checking $e(g, \sigma) = e(g^x, H(m))$.

[2] Dan Boneh and Matthew K. Franklin. "Identity-Based Encryption from the Weil Pairing". In: *CRYPTO*. vol. 2139. Lecture Notes in Computer Science. Springer, 2001, pp. 213–229.

BLS Signature [BLS04][3]

BLS Digital Signature

- Setup: (1) a cyclic multiplicative group $\mathbb{G}$ with a generator g of prime order q. (2) a cryptographic hash function $H : \{0,1\}^* \to \mathbb{G}$.
- Gen: select a random $x \in \mathbb{Z}_q$, secret key is x, public key is $y = g^x \in \mathbb{G}$.
- Sgn: a message $m \to$ a signature $\sigma = H(m)^x \in \mathbb{G}$.
- Ver: given a message m, a signature σ, a public key $y = g^x$, determine whether $\sigma = H(m)^{\log_g y}$ by checking if $e(g, \sigma) = e(y, H(m))$ holds.

[3] Dan Boneh, Ben Lynn, and Hovav Shacham. "Short Signatures from the Weil Pairing". In: *J. Cryptol.* 17.4 (2004), pp. 297–319.

BLS Signature [BLS04][3]

BLS Digital Signature

- Setup: (1) a cyclic multiplicative group $\mathbb{G}$ with a generator g of prime order q. (2) a cryptographic hash function $H : \{0,1\}^* \to \mathbb{G}$.
- Gen: select a random $x \in \mathbb{Z}_q$, secret key is x, public key is $y = g^x \in \mathbb{G}$.
- Sgn: a message $m \to$ a signature $\sigma = H(m)^x \in \mathbb{G}$.
- Ver: given a message m, a signature σ, a public key $y = g^x$, determine whether $\sigma = H(m)^{\log_g y}$ by checking if $e(g,\sigma) = e(y, H(m))$ holds.

Unforgeablility: the forging problem is a CDH problem. Given g, the public key $y = g^x$, and a message digest $H(m)$, it's hard to output a signature $\sigma = H(m)^{\log_g y}$.

[3] Dan Boneh, Ben Lynn, and Hovav Shacham. "Short Signatures from the Weil Pairing". In: *J. Cryptol.* 17.4 (2004), pp. 297–319.

Component 2: Shamir's secret sharing

(Threshold) secret sharing: share a secret x among n parties $\{P_1, P_2, \ldots, P_n\}$.

- Each party P_i knows a share of the secret x_i.
- Any subset of $\geq t$ parties can collaborate to reconstruct x using their shares.
- Any subset of $< t$ parties cannot reconstruct x.

Component 2: Shamir's secret sharing

(Threshold) secret sharing: share a secret x among n parties $\{P_1, P_2, \ldots, P_n\}$.

- Each party P_i knows a share of the secret x_i.
- Any subset of $\geq t$ parties can collaborate to reconstruct x using their shares.
- Any subset of $< t$ parties cannot reconstruct x.

Very simple construction: use a univariate polynomial $f(z) = a_0 + a_1 z + \cdots a_{t-1} z^{t-1}$ of degree $\leq t-1$.

- Choose f such that $f(0) = a_0 = x$, but other coefficients are uniformly at random.
- Send $f(i)$ to P_i, for $i \in \{1, 2, \ldots, n\}$.
- Reconstruction (Lagrange interpolation) using t shares $\{(j_k, x_{j_k})\}_{k=1}^{t}$:

$$f(z) = \sum_{k=1}^{t} \prod_{i \neq k} \frac{z - j_i}{j_k - j_i} \cdot x_{j_k} = \sum_{k=1}^{t} \lambda_{j_k}(z) x_{j_k}$$

BLS threshold signature [Bol03][4]

- Setup: same as BLS signature.
- Gen: (suppose there is a trusted dealer) choose a random $x \in \mathbb{Z}_q$ as the secret key, distribute the shares x_i to P_i. The public key is $y = g^x \in \mathbb{G}$.
- Sig: suppose $\{P_{j_1}, P_{j_2}, \ldots, P_{j_t}\}$ collaborate. Each P_{j_i} generates *partial signature* $\sigma_{j_i} = H(m)^{x_{j_i}}$. To compute $\sigma = H(m)^x$ (without revealing x to anyone):

$$\sigma = H(m)^{f(0)} = H(m)^{\sum_{i=1}^{t} \lambda_{j_i}(0) x_{j_i}} = \prod_{i=1}^{t} (H(m)^{x_{j_i}})^{\lambda_{S,j_i}(0)} = \prod_{i=1}^{t} \sigma_{j_i}^{\lambda_{S,j_i}(0)}$$

- Ver: same as BLS signature, using $\mathrm{sk} = y$: $e(g, \sigma) = e(y, H(m))$.

[4] Alexandra Boldyreva. "Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme". In: *Public Key Cryptography*. Vol. 2567. Lecture Notes in Computer Science. Springer, 2003, pp. 31–46.

BLS threshold signature [Bol03][4]

- Setup: same as BLS signature.
- Gen: (suppose there is a trusted dealer) choose a random $x \in \mathbb{Z}_q$ as the secret key, distribute the shares x_i to P_i. The public key is $y = g^x \in \mathbb{G}$.
- Sig: suppose $\{P_{j_1}, P_{j_2}, \ldots, P_{j_t}\}$ collaborate. Each P_{j_i} generates *partial signature* $\sigma_{j_i} = H(m)^{x_{j_i}}$. To compute $\sigma = H(m)^x$ (without revealing x to anyone):

$$\sigma = H(m)^{f(0)} = H(m)^{\sum_{i=1}^{t} \lambda_{j_i}(0) x_{j_i}} = \prod_{i=1}^{t} (H(m)^{x_{j_i}})^{\lambda_{S,j_i}(0)} = \prod_{i=1}^{t} \sigma_{j_i}^{\lambda_{S,j_i}(0)}$$

- Ver: same as BLS signature, using $\mathrm{sk} = y$: $e(g, \sigma) = e(y, H(m))$.

Very efficient: $O(1)$ signature size; $O(1)$ verification time. Gen: $O(t \log^2 t)$.

[4] Alexandra Boldyreva. "Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme". In: *Public Key Cryptography*. Vol. 2567. Lecture Notes in Computer Science. Springer, 2003, pp. 31–46.

Others

- **Schnorr threshold signature [KG20]** is also popular. It does not require bilinear pairing, but requires interaction among the set of signers.
- **(Threshold) Group/Ring signature [BSS02; RST01]**: any (subset of) member can generate a signature on behalf of the group, but no one knows which member generated it.
- **Static vs. Adaptive Security [BL22; CKM23]**: whether the adversary controls fixed $t-1$ nodes throughout the protocol, or can change the set of corrupted nodes.

Table of Contents

DKG to setup threshold signatures

In BLS threshold signature, we assume that a trusted dealer choooses a random x and distributes the shares among $\{P_1, \ldots, P_n\}$.

- the dealer might distribute wrong shares!

DKG to setup threshold signatures

In BLS threshold signature, we assume that a trusted dealer choooses a random x and distributes the shares among $\{P_1, \ldots, P_n\}$.

- the dealer might distribute wrong shares!
  - **verifiable** secret sharing: prove that shares/reconstruction are correct.

DKG to setup threshold signatures

In BLS threshold signature, we assume that a trusted dealer choooses a random x and distributes the shares among $\{P_1, \dots, P_n\}$.

- the dealer might distribute wrong shares!
  - **verifiable** secret sharing: prove that shares/reconstruction are correct.
- the dealer knows x. With x, he can create a signature by himself.

DKG to setup threshold signatures

In BLS threshold signature, we assume that a trusted dealer choooses a random x and distributes the shares among $\{P_1, \ldots, P_n\}$.

- the dealer might distribute wrong shares!
  - **verifiable** secret sharing: prove that shares/reconstruction are correct.
- the dealer knows x. With x, he can create a signature by himself.
  - Remove the dealer, use a distributed key generation (DKG) protocol.

DKG to setup threshold signatures

In BLS threshold signature, we assume that a trusted dealer choooses a random x and distributes the shares among $\{P_1, \ldots, P_n\}$.

- the dealer might distribute wrong shares!
  - **verifiable** secret sharing: prove that shares/reconstruction are correct.
- the dealer knows x. With x, he can create a signature by himself.
  - Remove the dealer, use a distributed key generation (DKG) protocol.

Intuition of DKG: nobody should know x, then let P_1 select s_1 and P_2 select s_2, define $x = s_1 + s_2$.

DKG to setup threshold signatures

In BLS threshold signature, we assume that a trusted dealer chooooses a random x and distributes the shares among $\{P_1, \ldots, P_n\}$.

- the dealer might distribute wrong shares!
  - **verifiable** secret sharing: prove that shares/reconstruction are correct.
- the dealer knows x. With x, he can create a signature by himself.
  - Remove the dealer, use a distributed key generation (DKG) protocol.

Intuition of DKG: nobody should know x, then let P_1 select s_1 and P_2 select s_2, define $x = s_1 + s_2$.

Issue: what if P_1 and P_2 collude with each other?

DKG to setup threshold signatures

In BLS threshold signature, we assume that a trusted dealer choooses a random x and distributes the shares among $\{P_1, \ldots, P_n\}$.

- the dealer might distribute wrong shares!
  - **verifiable** secret sharing: prove that shares/reconstruction are correct.
- the dealer knows x. With x, he can create a signature by himself.
  - Remove the dealer, use a distributed key generation (DKG) protocol.

Intuition of DKG: nobody should know x, then let P_1 select s_1 and P_2 select s_2, define $x = s_1 + s_2$.

Issue: what if P_1 and P_2 collude with each other?
Solution: let every P_i choose s_i, so that $x = s_1 + s_2 + \cdots s_n$. Every P_i shares s_i with others using VSS. [Pedersen'91]

Asynchronous DKG

Fault tolerant distributed protocols are expensive! Consider the cost of one node broadcasting a message to all nodes (n is the number of all nodes and t is the number of corrupted nodes):

- In synchronous networks (messages are delivered with known bounded delays) ($n = 2t + 1$): $O(n^2)$ communication.
- In asynchronous networks (messages are might be delayed arbitrarily long) ($n = 3t + 1$): $O(n^2)$ communication.

ADKG complexity of [Das+22][5]: $O(\kappa n^3)$ total communication, where κ is the output size of a cryptographic hash function.
ADKG might be the performance bottleneck of threshold signatures.

Recent research tries to improve ADKG, that ideally requires $O(n^2)$ communication.

[5] Sourav Das et al. "Practical Asynchronous Distributed Key Generation". In: SP. IEEE, 2022, pp. 2518–2534.

Table of Contents

Limitations of BLS threshold signature

- Unweighted: every member has the same unit weight.
  - In cryptocurrencies (Proof-of-Stake), accounts have (vastly) different weights.
  - Virtualization approach: suppose P_1 has weight 1 and P_2 has weight $10,000$, then P_2 should own $10,000$ secret shares in TS.

Limitations of BLS threshold signature

- Unweighted: every member has the same unit weight.
  - In cryptocurrencies (Proof-of-Stake), accounts have (vastly) different weights.
  - Virtualization approach: suppose P_1 has weight 1 and P_2 has weight $10,000$, then P_2 should own $10,000$ secret shares in TS.
- Fixed threshold: each polynomial (secret shares) corresponds to a fixed threshold t. For another threshold, we should setup another polynomial (and secret shares).

Succinct Non-interactive ARgument of Knowledge (Informal)

Verifiable computing: for some computation f, while computing/evaluating $f(x)$ given input x might take 1 year, given (x, y) verifying that $y = f(x)$ might take only 1 second.
SNARKs approach: firstly compile the computation to a circuit satisfiability problem (CSP).

Circuit Satisfiability Problem

Arithmetic circuit C.
Language $\mathcal{L}_C$: $\{x : \exists$ a witness w, such that $C(x, w) = 0\}$.
Relation $\mathcal{R}_C$: $\{(x, w) : C(x, w) = 0\}$.

$\mathcal{P}$ proves to $\mathcal{V}$ that $x \in \mathcal{L}_C$.

SNARKs (continued)

Polynomial commitment scheme: the witness w might be long. $\mathcal{P}$ represents w using a polynomial extension $\tilde{w}$, and only sends a $O(1)$ sized commitment of $\tilde{w}$ to $\mathcal{V}$. When $\mathcal{V}$ queries $\tilde{w}(r)$, $\mathcal{P}$ replies with the value and a proof, both succinct.
Interactive Oracle Proofs (for CSP): $\mathcal{P}$ and $\mathcal{V}$ interact a few round, $\mathcal{V}$ queries the polynomial committed by $\mathcal{P}$. While $\mathcal{P}$ might check $|C|$ constraints, $\mathcal{V}$ only checks a few ($O(1)$).
Non-interactive a proof can be reused for many different verifiers, that do not need to interact with the prover.

- Proof size: a succinct commitment c_w of w + a SNARK proof π. $|c_w|$ and π can be $O(1)$ field elements.
- Verification time: $O(1)$ field operation + $|x|$ to read the input.
- Prover time: ideally only slightly more than the time to evaluate the circuit.

Generic SNARK for weighted, multi-threshold TS

$\mathbf{pk} = [\mathrm{pk}_1, \mathrm{pk}_2, \ldots, \mathrm{pk}_n]$, $\mathbf{w} = [w_1, w_2, \ldots, w_n]$
The verifier agree and can access **pk**, **w** via succinct commitments.

Generic SNARK for weighted, multi-threshold TS

$\mathbf{pk} = [\mathrm{pk}_1, \mathrm{pk}_2, \ldots, \mathrm{pk}_n]$, $\mathbf{w} = [w_1, w_2, \ldots, w_n]$
The verifier agree and can access **pk**, **w** via succinct commitments.

A subset S of $\{P_1, \ldots, P_n\}$ generate partial signatures $\sigma_i = H(m)^{x_i}$.

Generic SNARK for weighted, multi-threshold TS

$\mathbf{pk} = [\mathrm{pk}_1, \mathrm{pk}_2, \ldots, \mathrm{pk}_n]$, $\mathbf{w} = [w_1, w_2, \ldots, w_n]$
The verifier agree and can access **pk**, **w** via succinct commitments.

A subset S of $\{P_1, \ldots, P_n\}$ generate partial signatures $\sigma_i = H(m)^{x_i}$.

CSP for WTS

Input for $\mathcal{V}$ to read: $x = (m, t, c_{\mathbf{pk}}, c_{\mathbf{w}}, c_S, c_{\sigma_S})$
The witness: $w = (\mathbf{pk}, \mathbf{w}, S, \sigma_S)$
The relation $\mathcal{R}_{WTS}$:

$$\begin{gathered}
\mathbf{pk} \in \mathbb{G}^n;\ c_{\mathbf{pk}} = \mathrm{commit}(\mathbf{pk}) \\
\mathbf{w} \in \mathbb{F}^n, ||\mathbf{w}||_1 < |\mathbb{F}|;\ c_{\mathbf{w}} = \mathrm{commit}(\mathbf{w}) \\
S \subseteq \{1, 2, \ldots, n\};\ c_S = \mathrm{commit}(S) \\
\sigma_S \in \mathbb{G}^{|S|};\ c_{\sigma_S} = \mathrm{commit}(\sigma_S) \\
\forall i \in S, e(H(m), \mathrm{pk}_i) = e(\sigma_i, g) \\
t = \textstyle\sum_{i \in S} w_i
\end{gathered}$$

Specialized SNARK for weighted, multi-threshold TS [Das+23][6]

Special SNARKs Generic SNARKs are designed for a general class of problems (arithmetic circuit satisfiability problem/R1CS). They might not be optimal for a particular class of problems.
Inner Product Argument (IPA): prove that $\langle \mathbf{a}, \mathbf{b} \rangle = c$, $\mathbf{a}, \mathbf{b} \in \mathbb{F}^n$. Verifiers can access the commitment of **a** and **b**.

[6] Sourav Das et al. "Threshold Signatures from Inner Product Argument: Succinct, Weighted, and Multi-threshold". In: *CCS*. ACM, 2023, pp. 356–370.

Specialized SNARK for weighted, multi-threshold TS [Das+23][6]

Special SNARKs Generic SNARKs are designed for a general class of problems (arithmetic circuit satisfiability problem/R1CS). They might not be optimal for a particular class of problems.
Inner Product Argument (IPA): prove that $\langle \mathbf{a}, \mathbf{b} \rangle = c$, $\mathbf{a}, \mathbf{b} \in \mathbb{F}^n$. Verifiers can access the commitment of **a** and **b**.

- Let $\mathbf{b} = [b_1, b_2, \ldots, b_n] \in \{0,1\}^n$, $b_i = 1$ if P_i generates a partial signature $\sigma_i = H(m)^{x_i}$.
- The aggregate signature $\sigma_{\mathbf{b}} = \prod_{b_i=1} \sigma_i = H(m)^{\langle \mathbf{x}, \mathbf{b} \rangle}$. Verification key is $\prod_{b_i=1} y_i = g^{\langle \mathbf{x}, \mathbf{b} \rangle}$.
  - However, the aggregator should not learn the secret keys x_i. So they use general IPA for $\langle \sigma, \mathbf{b} \rangle$ and $\langle \mathbf{y}, \mathbf{b} \rangle$. One vector is in the field $\mathbb{F}$, the other in the group $\mathbb{G}$.
- The total weight is $\langle \mathbf{w}, \mathbf{b} \rangle$.

Result : the prover complexity is more practical.

[6] Sourav Das et al. "Threshold Signatures from Inner Product Argument: Succinct, Weighted, and Multi-threshold". In: *CCS*. ACM, 2023, pp. 356–370.

Table of Contents

Topic 1: succinct aggregate signatures and democratic voting

Aggregate Signatures n members may sign different messages. P_i creates a signature for m_i: $H(m)^{x_i}$.
How to aggregate these signatures to one short proof?

Topic 1: succinct aggregate signatures and democratic voting

Aggregate Signatures n members may sign different messages. P_i creates a signature for m_i: $H(m)^{x_i}$.
How to aggregate these signatures to one short proof?

Democratic voting Suppose the group votes for a leader and there are multiple candidates.
A candidate wins if he receives the most votes, not necessarily $> 50\%$.
A proof of candidate winning with 40% votes should show that all other candidates receives $< 40\%$ votes.

Topic 1: succinct aggregate signatures and democratic voting

Aggregate Signatures n members may sign different messages. P_i creates a signature for m_i: $H(m)^{x_i}$.
How to aggregate these signatures to one short proof?

Democratic voting Suppose the group votes for a leader and there are multiple candidates.
A candidate wins if he receives the most votes, not necessarily $> 50\%$.
A proof of candidate winning with 40% votes should show that all other candidates receives $< 40\%$ votes.

Generic SNARK works, maybe special SNARKs can make further improvement.

Topic 2: incremental DKG

If only a few new members join or a few old members leave, can we do better than rerun the complete DKG protocol again (the total communication cost is $O(\kappa n^3)$)?

Topic 3: alternative multi-threshold cryptosystem

Current (BLS) threshold signatures are based on signatures where verification result is binary.
A signature is either valid or invalid.

What if the verification step returns a value in range $[0, 1]$?
Discrete log cryptography might not achieve this. What about lattice cryptography, or another family of hard problems?

Summary

- BLS threshold signature, based on BLS signature and secret sharing.
- Weighted, multi-threshold TS, based on generic/special SNARKs.
- DKG, efficient asynchronous DKG, incremental DKG.
- Alternative multi-threshold cryptosystem.

Reference I

[APS18] Alessandro Amadori, Federico Pintore, and Massimiliano Sala. "On the discrete logarithm problem for prime-field elliptic curves". In: *Finite Fields Their Appl.* 51 (2018), pp. 168–182.

[BF01] Dan Boneh and Matthew K. Franklin. "Identity-Based Encryption from the Weil Pairing". In: *CRYPTO*. Vol. 2139. Lecture Notes in Computer Science. Springer, 2001, pp. 213–229.

[BL22] Renas Bacho and Julian Loss. "On the Adaptive Security of the Threshold BLS Signature Scheme". In: *CCS*. ACM, 2022, pp. 193–207.

[BLS04] Dan Boneh, Ben Lynn, and Hovav Shacham. "Short Signatures from the Weil Pairing". In: *J. Cryptol.* 17.4 (2004), pp. 297–319.

[Bol03] Alexandra Boldyreva. "Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme". In: *Public Key Cryptography*. Vol. 2567. Lecture Notes in Computer Science. Springer, 2003, pp. 31–46.

[BSS02] Emmanuel Bresson, Jacques Stern, and Michael Szydlo. "Threshold Ring Signatures and Applications to Ad-hoc Groups". In: *CRYPTO*. Vol. 2442. Lecture Notes in Computer Science. Springer, 2002, pp. 465–480.

Reference II

[CKM23] Elizabeth C. Crites, Chelsea Komlo, and Mary Maller. "Fully Adaptive Schnorr Threshold Signatures". In: *CRYPTO (1)*. Vol. 14081. Lecture Notes in Computer Science. Springer, 2023, pp. 678–709.

[Das+22] Sourav Das et al. "Practical Asynchronous Distributed Key Generation". In: *SP*. IEEE, 2022, pp. 2518–2534.

[Das+23] Sourav Das et al. "Threshold Signatures from Inner Product Argument: Succinct, Weighted, and Multi-threshold". In: *CCS*. ACM, 2023, pp. 356–370.

[KG20] Chelsea Komlo and Ian Goldberg. "FROST: Flexible Round-Optimized Schnorr Threshold Signatures". In: *SAC*. Vol. 12804. Lecture Notes in Computer Science. Springer, 2020, pp. 34–65.

[RST01] Ronald L. Rivest, Adi Shamir, and Yael Tauman. "How to Leak a Secret". In: *ASIACRYPT*. Vol. 2248. Lecture Notes in Computer Science. Springer, 2001, pp. 552–565.