Multi And Threshold Signatures for Starknet

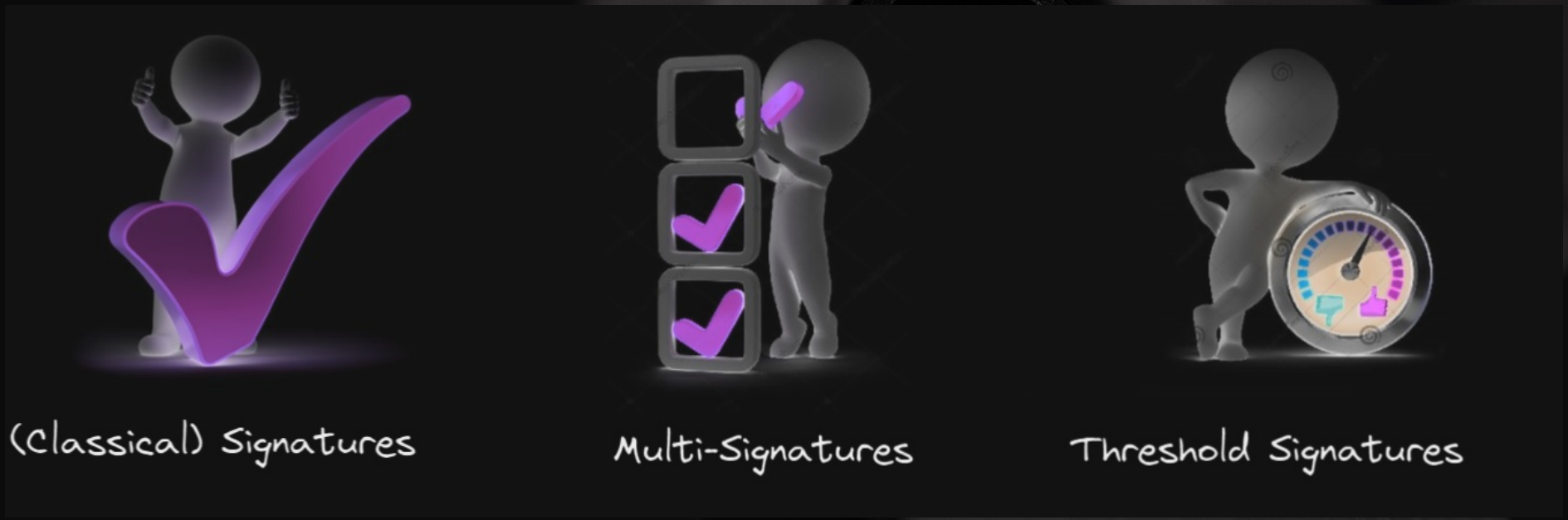
(Warming up for Lisbonn Hackaton)

Renaud Dubois

Ledger
Innovation Team

October 12, 2022

Summary


(Classical) Signatures
Multi-Signatures
Threshold Signatures

Summary

1. Signatures, MultiSigs and ThresholdSigs
   - Basic Concepts
   - Under the hood
2. Use cases
   - Multi factor authentication
   - Voting system
3. Ledger/Starknet Musig2, call for Lisbonn

Signatures

A digital signature is a mathematical scheme for verifying the authenticity of digital messages or documents.

Definition ((Classical) Digital Signature)

A signature scheme is a tuple of function:

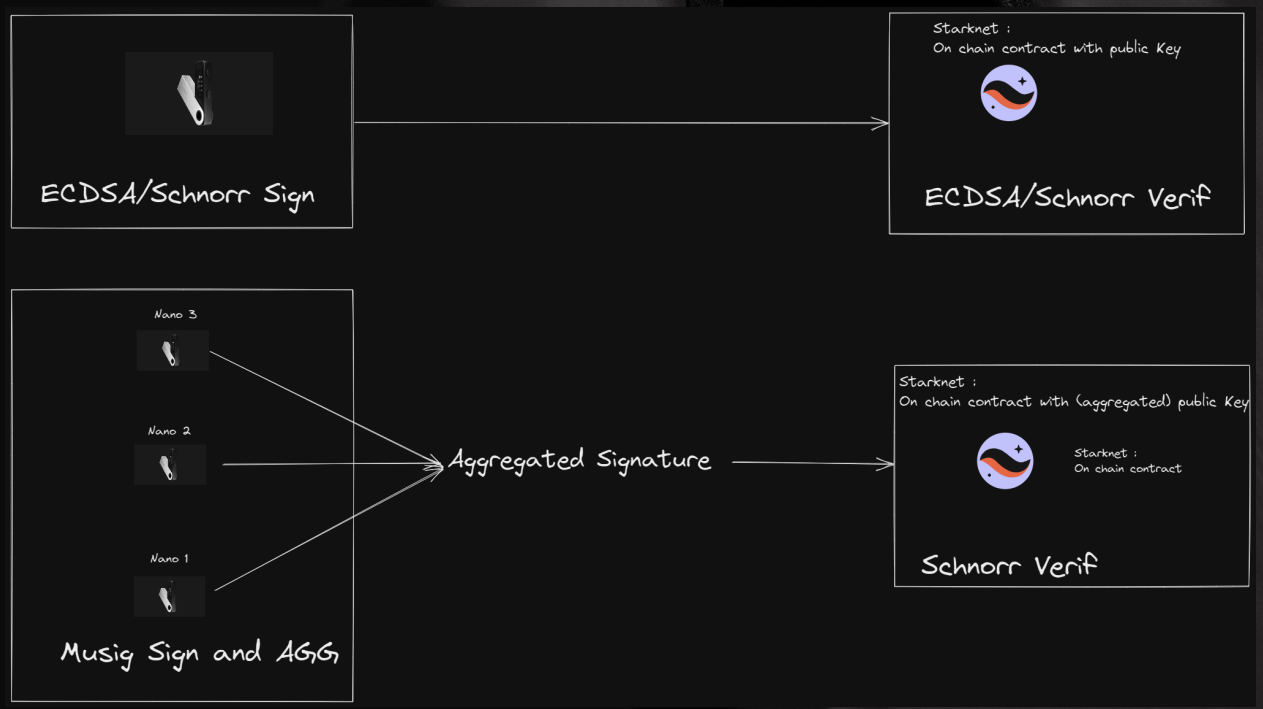
- $Setup$
- $KeyGen$
- $Sign$
- $Verify$

Most commonly used signature scheme is ECDSA (Bitcoin, Ethereum)

- implemented in Starknet/Cairo (P256, NTT/Stark friendly Starknet Curve)
- available in your favorite sdk Ledger

Multi-signatures

A multi-signature is a digital signature allowing users to *aggregate* their keys in an aggregated public key. The signatures are also aggregated. Verifier API is unchanged.



Multi-signatures

A multi-signature is a digital signature allowing users to *aggregate* their keys in an aggregated public key. The signatures are also aggregated. Verifier API is unchanged.

Definition ((Classical) Digital Signature)

A multisig scheme is a tuple of function:

- $(Setup, Keygen, Sign)$
- $KeyAgg$
- $SignAgg$
- $Verify$

Multi-signatures

A multi-signature is a digital signature allowing users to *aggregate* their keys in an aggregated public key. The signatures are also aggregated. Verifier API is unchanged.

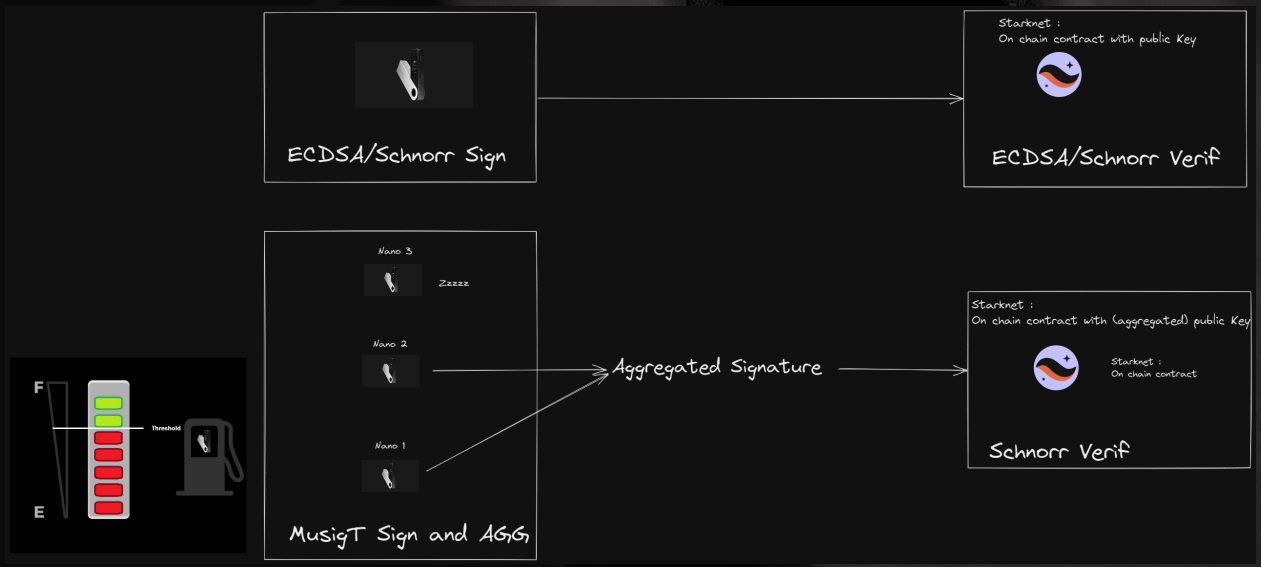
Advantages (over naive concatenation/trusted aggregator):

- only one signature over channel (bandwidth consumption)
- no need for a trusted aggregator (no remote private key, own your crypto !)
- no risk of contract failure (don't trust, no don't)
- verifier doesn't need to know the underlying group of users

Example: Bitcoin Taproot, BIP340

Threshold-signatures

A (k, n) threshold signature is a digital signature allowing a subset (threshold) of k users from n to *aggregate* a signature .



Threshold-signatures

A (k, n) threshold signature is a digital signature allowing a subset (threshold) of k users from n to *aggregate* a signature .

- $(Setup, Keygen, Sign)$
- $KeyAgg$
- $SignAgg$
- $Verify$

Advantages (over naive concatenation/trusted aggregator):

- only one signature over channel (bandwidth consumption)
- no need for a trusted aggregator (no remote private key, own your crypto !)
- no risk of contract failure (don't trust, no don't)
- verifier doesn't need to know the underlying group of users

Example: FROST(Blockstream).

EC-Schnorr and ECDSA

Definition (Common SetUp and KeyGen)

SetUp() Pick a curve with parameters (p, a, b, Gx, Gy, q) (weierstrass equations and formulaes).

KeyGen() Randomly select private key $x \leftarrow^{\$} F_q$, publish public key $Q = xG$ (flip x if Gy is even for xonly implementation).

Definition (ECDSA-Sign)

- $r = (kG)_x$
- $e = H(M)$
- $s = k^{-1}(e + sr)$
- $Sig = (R, s)$

EC-Schnorr and ECDSA

Definition (Common SetUp and KeyGen)

SetUp() Pick a curve with parameters (p, a, b, Gx, Gy, q) (weierstrass equations and formulaes).

KeyGen() Randomly select private key $x \leftarrow^{\$} F_q$, publish public key $Q = xG$ (flip x if Gy is even for xonly implementation).

Definition (Schnorr-Sign)

- $R = kG$
- $e = H(R||m)$
- $s = k - xe$
- $Sig = (R, s)$

EC-Schnorr and ECDSA

Definition (Common SetUp and KeyGen)

SetUp() Pick a curve with parameters (p, a, b, Gx, Gy, q) (weierstrass equations and formulaes).

KeyGen() Randomly select private key $x \leftarrow^{\$} F_q$, publish public key $Q = xG$ (flip x if Gy is even for xonly implementation).

Definition (ECDSA-Verif)

- $e = H(M)$
- $r' = (es^{-1}G + rs^{-1}Q)_x$
- $Accept\ if f.r == r'$

EC-Schnorr and ECDSA

Definition (Common SetUp and KeyGen)

SetUp() Pick a curve with parameters (p, a, b, Gx, Gy, q) (weierstrass equations and formulaes).

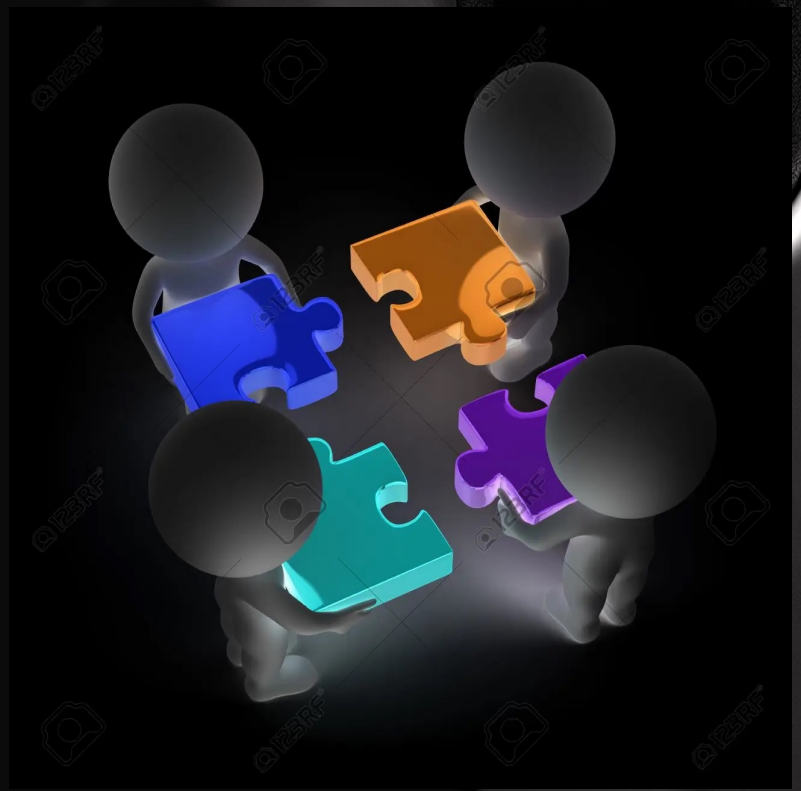
KeyGen() Randomly select private key $x \leftarrow^{\$} F_q$, publish public key $Q = xG$ (flip x if Gy is even for xonly implementation).

Definition (Schnorr-Verif)

- $e = H(M)$
- $R' = sG + eQ$
- $Accept\ if f.e' == H(R'||m)$

Musig2: using Schnorr additive properties

Schnorr is linear in (k, x), while ECDSA has degree two monomial, and linerarity is good.
Linearity allow homomorphic additions.
Idea: split X into $X = \sum a_i X_i$, k into $k = \sum k_i$.



Musig2: using Schnorr additive properties

Operation	Schnorr	Insec_Musig
KeyGen	$X = xG$	$X_i = x_iG$
KeyAgg	-	$X = \sum_{i=0}^{n-1} a_i X_i$
Nonce	r	k_i
Ephemeral	$R = rG$	$R_i = k_iG$
Aggregate R	-	$R = (\sum_{i=0}^{n-1} a_i.k_i).G = k.G$
Hash	$e = H(m\|\|R)$	$e = H(m\|\|R)$
Sign	$s = k - xe$	$s_i = k_i - a_i x_i e$
Aggregate s	-	$s = \sum s_i = k - xe$

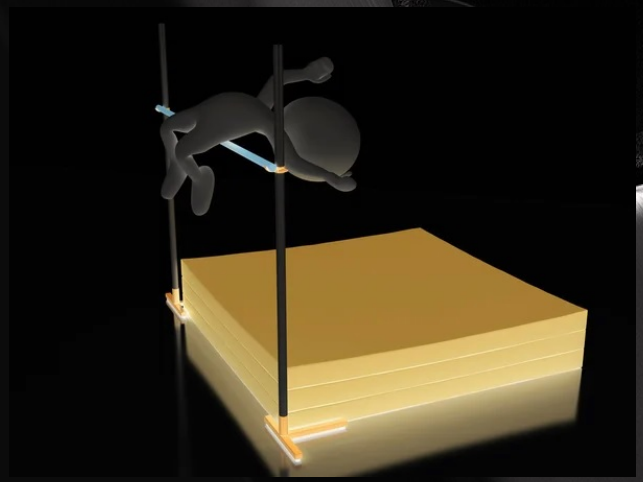
Musig2: using Schnorr additive properties

Musig2 uses a vectorial nonce of length μ, injected in previous Insec_Musig scheme.

Operation	Schnorr	Musig2
KeyGen	$X = xG$	$X_i = x_i G$
KeyAgg	-	$X = \sum_{i=0}^{n-1} a_i X_i$
Nonce	r	$\vec{r_i} = (r_{i1}, \ldots, r_{i\mu})$
Ephemeral	$R = rG$	$\vec{R_i} = \vec{r_i} G$
Hash Nonce	-	$b = H(X \|\| R_0 \ldots R_\mu \|\| m)$
Aggregate R	-	$R = \sum_{j=1}^{\mu} b^{j-1} (\sum_{i=0}^{n-1} a_i.k_i).G = k.G$
Hash	$e = H(m \|\| R)$	$e = H(m \|\| R)$
Sign	$s = k - xe$	$s_i = (\sum_{j=1}^{\mu} k_i j b^{j-1}) - a_i x_i e$
Aggregate s	-	$s = \sum s_i = k - xe$

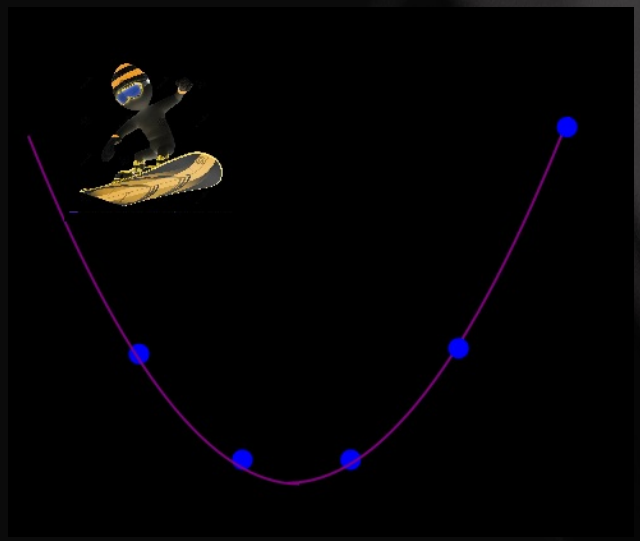
Musig2: Thresholdisation Principle

Thresholdisation use the principle of Shamir's secret sharing scheme , which is in fact a reed solomon error correcting code.
Goal: Given enough shares, it is possible to reconstruct the initial value.



Musig2: Thresholdisation Principle

Lagrange interpolation enables to switch from points to polynomial coefficients using the following formulaes:



$$l_j(x) = \prod_{m \neq j} \frac{x - x_m}{x_j - x_m}.$$

$$L(x) = \sum_{j=0}^{k} P(x_j) l_j(x).$$

The transformation L from $(P_0 \dots P_k)$ to $(a_0 \dots a_k)$ is a linear transformation in x.

Musig2: Thresholdisation Principle

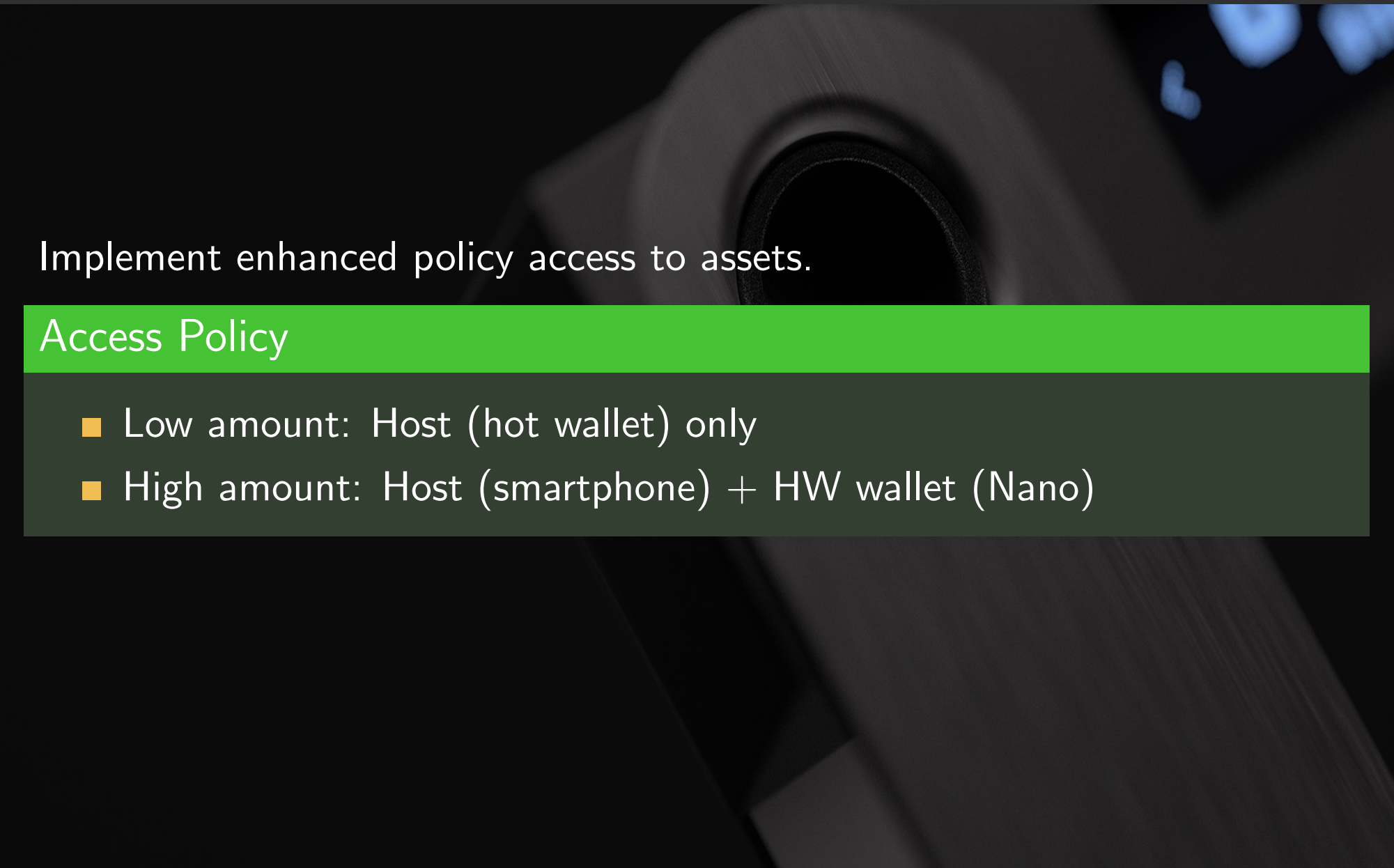
Key ideas:

- interprete aggregated secret key as a polynomial P of degree k,
- each share (user secret key) is a point of the polynomial,
- blind the computation in the curve domain to perform the aggregation only handling public elements,
- replace '$\sum_{i=0}^{n}$' in previous scheme by Lagrange polynomials,
- some more steps are necessary (commitments) to avoid cheating.

Read FROST for full description.

Sidenote: This is closely related to the principle of FRI used in starks.

Multi factor authentication to Starknet Contract

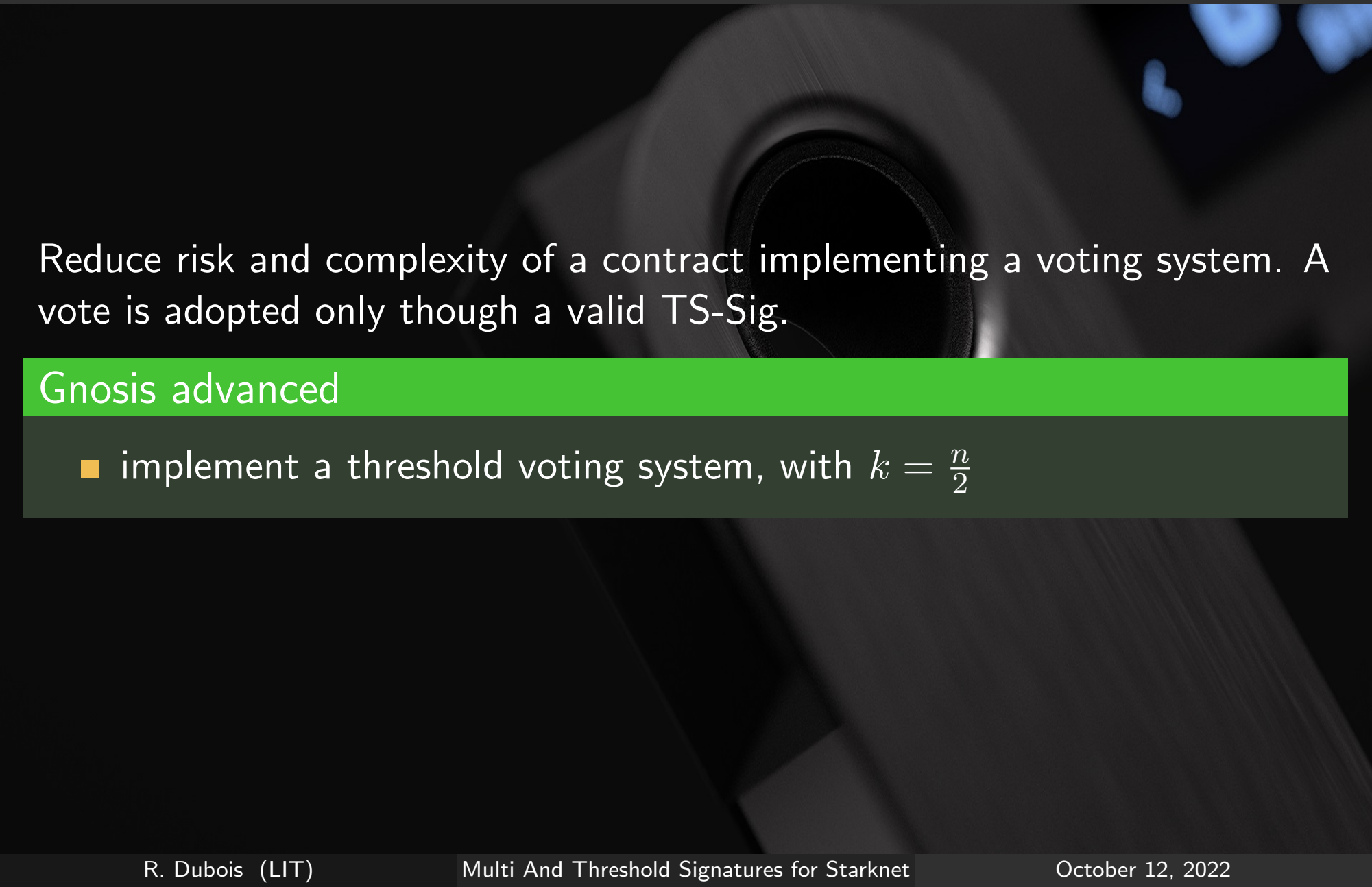


Implement enhanced policy access to assets.

Access Policy

- Low amount: Host (hot wallet) only
- High amount: Host (smartphone) + HW wallet (Nano)

Voting system



Reduce risk and complexity of a contract implementing a voting system. A vote is adopted only though a valid TS-Sig.

Gnosis advanced

- implement a threshold voting system, with $k = \frac{n}{2}$

Ledger/Starknet Musig2

Current state:

- Schnorr verification available in Cairo, (100%)
- High-level simulation in Sagemath of full protocol (Sign/Verify for a pool of users) (100%)
- Musig2 implementation on top of a virtualization layer (only integrating bolos for now) (still some issues)

C Library (Nano Signer)

SCAN ME

Cairo Code (Contract Verifier)

SCAN ME

Starknet Hackaton

Join the team for:

- Front end integration (wallet, current development over Argent) over verifier (Cairo) or signer (C)
- Integration of a different accelerator/library in the virtualization layer (C)
- Contribute to the threshold version (Sagemath/C)

Questions ?

C Library (Nano Signer)

SCAN ME

Cairo Code (Contract Verifier)

SCAN ME