Hacky Schnorr for MPC and ZK with applications to Kohaku wallet

Simon Masson, Renaud Dubois
ZKNox

Privacy and Compliance Summit

November 18th, 2025 – Buenos Aires, DevConnect

ZKNOX team

Nicolas Bacca
20^{+} years experience (10^{+}y web3)
Security and hardware specialist
Prev. Ledger cofounder/CTO

Renaud Dubois
20^{+} years experience (3^{+}y web3)
Cryptographer
Prev. Ledger, Thales

Simon Masson
8^{+} years experience (4^{+}y web3)
Cryptographer
Prev. Heliax, Thales

ZKNOX team

Nicolas Bacca
20^+ years experience (10^+y web3)
Security and hardware specialist
Prev. Ledger cofounder/CTO

Renaud Dubois
20^+ years experience (3^+y web3)
Cryptographer
Prev. Ledger, Thales

Simon Masson
8^+ years experience (4^+y web3)
Cryptographer
Prev. Heliax, Thales

Expertise and innovation to every challenge on the whole security chain:

- user end
  (secure enclaves, hardware wallets),
- back end
  (TEE, HSMs),
- on-chain
  (smart contracts).

ZKNOX team

Nicolas Bacca
20^+ years experience (10^+y web3)
Security and hardware specialist
Prev. Ledger cofounder/CTO

Renaud Dubois
20^+ years experience (3^+y web3)
Cryptographer
Prev. Ledger, Thales

Simon Masson
8^+ years experience (4^+y web3)
Cryptographer
Prev. Heliax, Thales

Expertise and innovation to every challenge on the whole security chain:

- user end (secure enclaves, hardware wallets),
- back end (TEE, HSMs),
- on-chain (smart contracts).

`https://zknox.eth.limo/`

`https://github.com/zknoxhq/`

Summary

Signatures

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

Signatures

Definition ((Classical) Digital Signature)

A signature scheme is a tuple of function:

- *Setup*: returns domain parameters $E(F_p), G, H$
- $KeyGen(E(F_p), G, H, seed)$: returns $(pvk, pubk) = (x, Q)$
- $Sign(x, message)$: returns Sig
- $Verify(Sig, Q)$: returns `true/false`

Formulaes for elliptic computations.
Dictionnary of curves parameters

Signatures

Properties

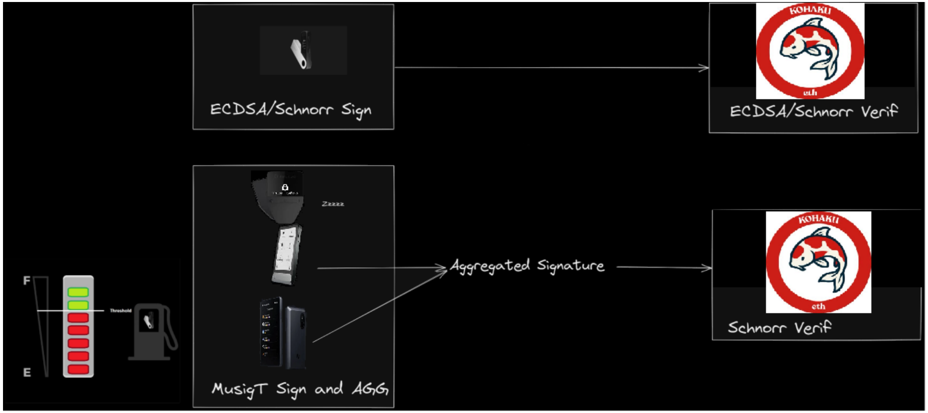
- Unforgeable
- Non repudiation
- *Not reusable*

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

- A painfull patent prevented Schnorr from being used, now expired
- Schnorr is used in Taproot, Ed25519, EDDSA POSEIDON (CIRCOMLIB, RAILGUN)

Threshold-signatures

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



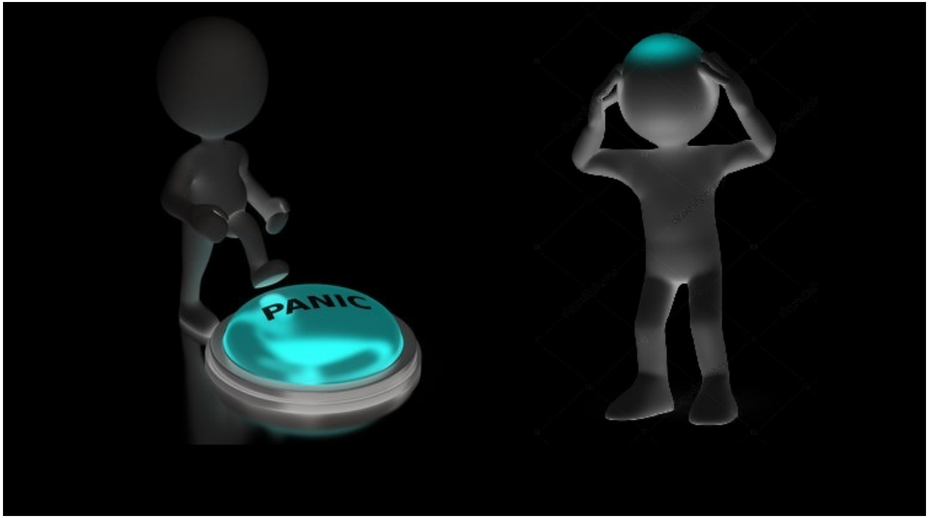
Threshold-signatures

Definition (Threshold Signatures)

A multisig scheme is a tuple of function:

- $(Setup, Verify, Sign)$
- $DistributedKeygen$,
- $KeyAgg(Q_1, \dots Q_n)$ returns X
- $SignAgg(Sig_1, \dots, Sig_n)$ returns Sig

Disclaimer



EC-Schnorr and ECDSA

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

Operation	Schnorr	ECDSA
KeyGen	$Q = xG$	$Q = xG$
Nonce*	k	k
Ephemeral	$R = kG$	$R = kG$
Hash	$e = H(m\|\|R)$	$e = H(m)$
Sign	$s = k - xe$	$s = k^{-1}(e + xr)$
	$Sig = (R, s)$	$Sig = (r, s)$
Verif	$R' = sG + eQ$	$r' = (es^{-1}G + rs^{-1}Q)_x$
	Accept if R'=R	Accept if r'=r

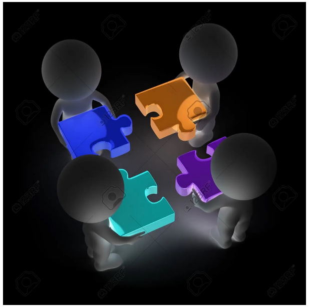
(nonce generation may use RFC6979 for misuse resistance)*

Musig2: using Schnorr additive properties

Schnorr s part is linear in (k, x) and linerarity is cool:

$$\begin{aligned} s(k, x_1) + s(k, x_2) &= s(k, x), \quad \forall x = x_1 + x_2 \\ s(k_1, x) + s(k_2, x) &= s(k, x), \quad \forall k = k_1 + k_2 \end{aligned}$$

(while ECDSA has degree two monomial in (k, x))



Musig2: using Schnorr additive properties

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	$X = xG$	$X_i = x_i G$
KeyAgg	-	X$=\sum_{i=0}^{n-1} a_i X_i$
Nonce*	k	k_i
Ephemeral	$R = kG$	$R_i = k_i G$
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	-	$s = \sum s_i = k - xe$

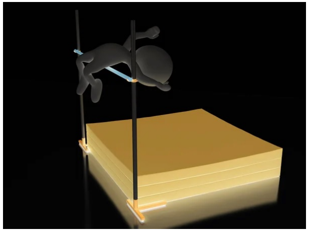
Musig2: using Schnorr additive properties

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	-	$\mathrm{X}{=}\sum_{i=0}^{n-1} a_i X_i$
Nonce*	k	$\vec{k_i} = (k_{i1}, \ldots, k_{i\mu})$
Ephemeral	$R = kG$	$\vec{R_i} = \vec{k_i} G$
Hash Nonce	-	$b = H(X \vert\vert R_0 \ldots R_\mu \vert\vert m)$
Aggregate R	-	$R = \sum_{j=1}^{\mu} b^{j-1} (\sum_{i=0}^{n-1} a_i.k_i).G = k.G$
Hash	$e = H(m \vert\vert R)$	$e = H(m \vert\vert R)$
Sign	$s = k - xe$	$s_i = (\sum_{j=1}^{\mu} k_i j b^{j-1}) - a_i x_i e$
Aggregate s	-	$s = \sum s_i = k - xe$

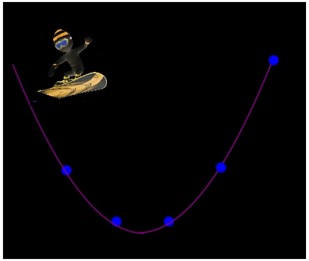
Musig2: Thresholdisation Principle

Thresholdisation use the principle of Shamir's secret sharing scheme , which is in fact a reed solomon erasure 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 $(P_0 \ldots P_k)$ to $(a_0 \ldots a_k)$ is a linear transformation in x.

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

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.

Hacky Mul for secp256k1

- Point Multiplication is expensive in solidity (best implementation to date, FCL: 69K with huge precomputations, 200K otherwise). https://eprint.iacr.org/2023/939.pdf
- How to have a low gas Schnorr ?

Hacky Mul for secp256k1

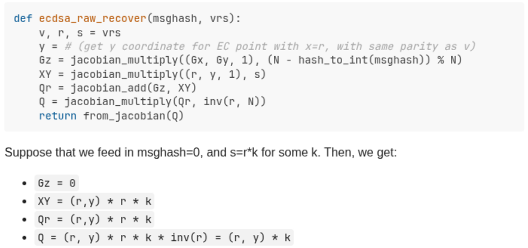
- Point Multiplication is expensive in solidity (best implementation to date, FCL: 69K with huge precomputations, 200K otherwise). https://eprint.iacr.org/2023/939.pdf
- How to have a low gas Schnorr ?

Use a ZK verification: complex and expensive, but available in RAILGUN for the privacy property, not only computation.

Hacky Mul for secp256k1

- Point Multiplication is expensive in solidity (best implementation to date, FCL: 69K with huge precomputations, 200K otherwise). https://eprint.iacr.org/2023/939.pdf
- How to have a low gas Schnorr ?

Original idea for k1 from the V:



```
def ecdsa_raw_recover(msghash, vrs):
    v, r, s = vrs
    y = # (get y coordinate for EC point with x=r, with same parity as v)
    Gz = jacobian_multiply((Gx, Gy, 1), (N - hash_to_int(msghash)) % N)
    XY = jacobian_multiply((r, y, 1), s)
    Qr = jacobian_add(Gz, XY)
    Q = jacobian_multiply(Qr, inv(r, N))
    return from_jacobian(Q)
```

Suppose that we feed in msghash=0, and s=r*k for some k. Then, we get:

- `Gz = 0`
- `XY = (r,y) * r * k`
- `Qr = (r,y) * r * k`
- `Q = (r, y) * r * k * inv(r) = (r, y) * k`

Hacky Mul for secp256r1

Inspired both by V and Y. El Housny: hinted mul for secp256r1 using 7951 verify;
We want to check that given α, Q the equality $alpha.G = Q$ holds.
ECDSA verification:

$$(i, j) = (h.s^-1).G + (r.s^-1).Q$$

$$return \quad i == r$$

To check that provided result (hint) $Q = \alpha.G$:

$$(h, r, s, Q) = (1 - \alpha)x, x, -x, Q$$

Where x is G x-ordinate.
if $ecdsa\_verify(h, r, s, q) = true$, then provided hint is correct.