FROST and MuSig2: From Theory to Practice

Ian Goldberg
University of Waterloo

Chelsea Komlo
University of Waterloo, SandboxAQ

Jonas Nick
Blockstream

Tim Ruffing
Blockstream

Yannick Seurrin
Ledger

November 22, 2024

Why a shared prize between FROST and MuSig2?

A Story in Four Parts

Cryptology ePrint Archive

Paper 2020/852

FROST: Flexible Round-Optimized Schnorr Threshold Signatures

Chelsea Komlo and Ian Goldberg

Abstract

Unlike signatures in a single-party setting, threshold signatures require cooperation among a threshold number of signers each holding a share of a common private key. Consequently, generating signatures in a threshold setting imposes overhead due to network rounds among signers, proving costly when secret shares are stored on network-limited devices or when coordination occurs over unreliable networks. In this work, we present FROST, a Flexible Round-Optimized Schnorr Threshold signature scheme that reduces network overhead during signing operations while employing a novel technique to protect against forgery attacks applicable to similar schemes in the literature. FROST improves upon the state of the art in Schnorr threshold

FROST and MuSig2

musig2-...7.pdf

TR

Tim Ruffing<crypto@timruffing.de>

To: Chelsea Komlo; Ian Goldberg

Cc: Jonas Nick <jonasdnick@gmail.com>; Yannick Seurin <yannick.seurin@m4x.org>

musig2-2020-07-17.pdf
643 KB

Hey Chelsea and Ian,

I hope you're enjoying PETS. :) I'm not registered for PETS but Ian, just today I watched your welcome note on YouTube. I immediately told Pedro to submit to PETS with his students to get a chain of presenters "Goldberg -> Kate -> Moreno-Sanchez -> X". :-P

A few days ago we saw that you uploaded a new revision of FROST to ePrint. We had a look at your paper and it turns out that in the past months we've been working concurrently on a paper ("MuSig2") that has a somewhat different goals but shares some ideas. In particular, we independently came up with the idea to use a linear combination of two nonces in order to obtain a two-round scheme which offers security in parallel sessions (Chelsea, I mentioned in a Twitter chat back in March that we're working on a new technique for a two-round scheme, this was what I was talking about.)

Cryptology ePrint Archive

Paper 2020/1261

MuSig2: Simple Two-Round Schnorr Multi-Signatures

Jonas Nick, Blockstream
Tim Ruffing, Blockstream
Yannick Seurin, ANSSI

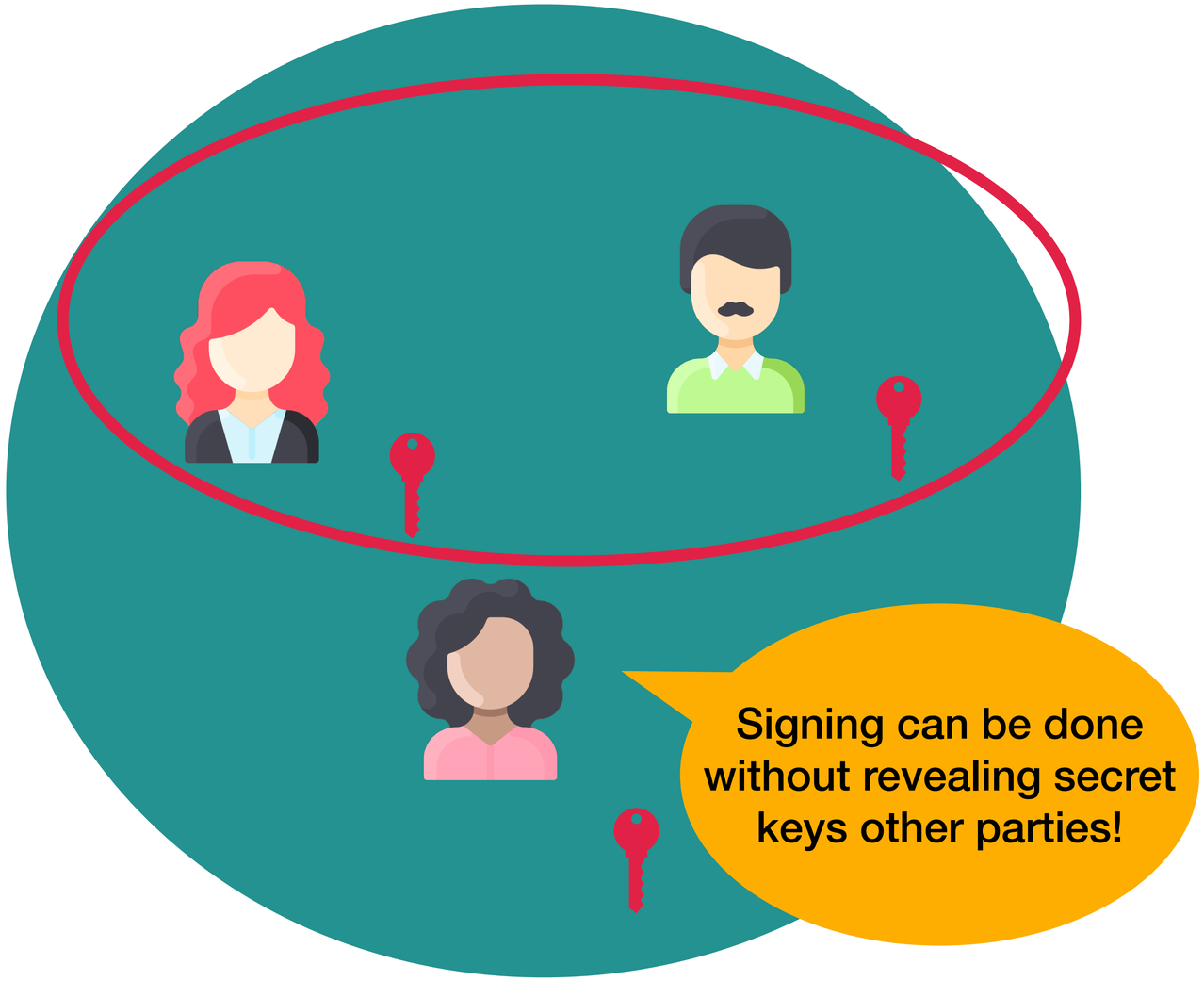
Abstract

Multi-signatures enable a group of signers to produce a joint signature on a joint message. Recently, Drijvers et al. (S&P'19) showed that all thus far proposed two-round multi-signature schemes in the pure DL setting (without pairings) are insecure under concurrent signing sessions. While Drijvers et al. proposed a secure two-round scheme, this efficiency in terms of rounds comes with the price of having signatures that are more than twice as large as Schnorr signatures, which are becoming popular in cryptographic systems due to their practicality (e.g., they will likely be adopted in Bitcoin). If one

Summary: Science is Magic

Theory

What are Threshold Signatures?



Public Key

- Ideally t-out-of-n
- Key generation via trusted dealer or DKG
- Secure up to (t-1) corruptions

(2,3) Example

What are Multi-Signatures? [IN83, BN06]

Public Key

(3,3) Example

- n-out-of-n
- $t = n - 1$
- key aggregation to produce PK
- n signers can be spontaneous

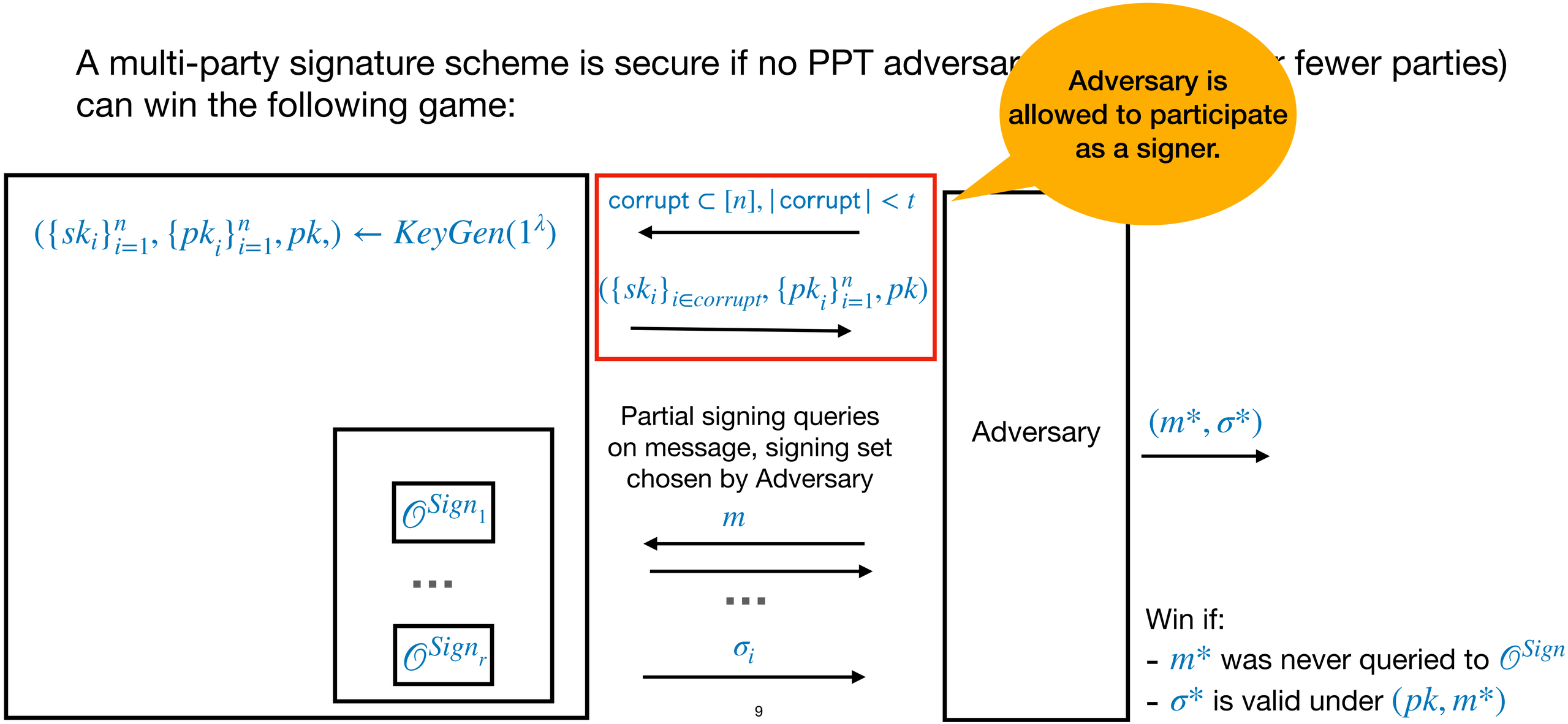
Security of Multi-Party Signatures

Unforgeability:
Attacker cannot forge signatures.

Liveness:
System can always create signatures.

Unforgeability

A multi-party signature scheme is secure if no PPT adversar fewer parties) can win the following game:



(Single-Party) Schnorr Signatures

$\sigma = (R, z)$

To generate a key pair:

$sk \xleftarrow{\$} \mathbb{Z}_q \; ; \; PK \leftarrow g^{sk}$

To sign a message m:

$r \xleftarrow{\$} \mathbb{Z}_q \; ; \; R \leftarrow g^r$

$c \leftarrow H(PK, m, R)$

$z \leftarrow r + csk$

To verify (PK, σ, m):

$c \leftarrow H(PK, m, R)$

$R \cdot PK^c \stackrel{?}{=} g^z$

output accept/reject

Multi-Party Schnorr Signatures

How to share r ?

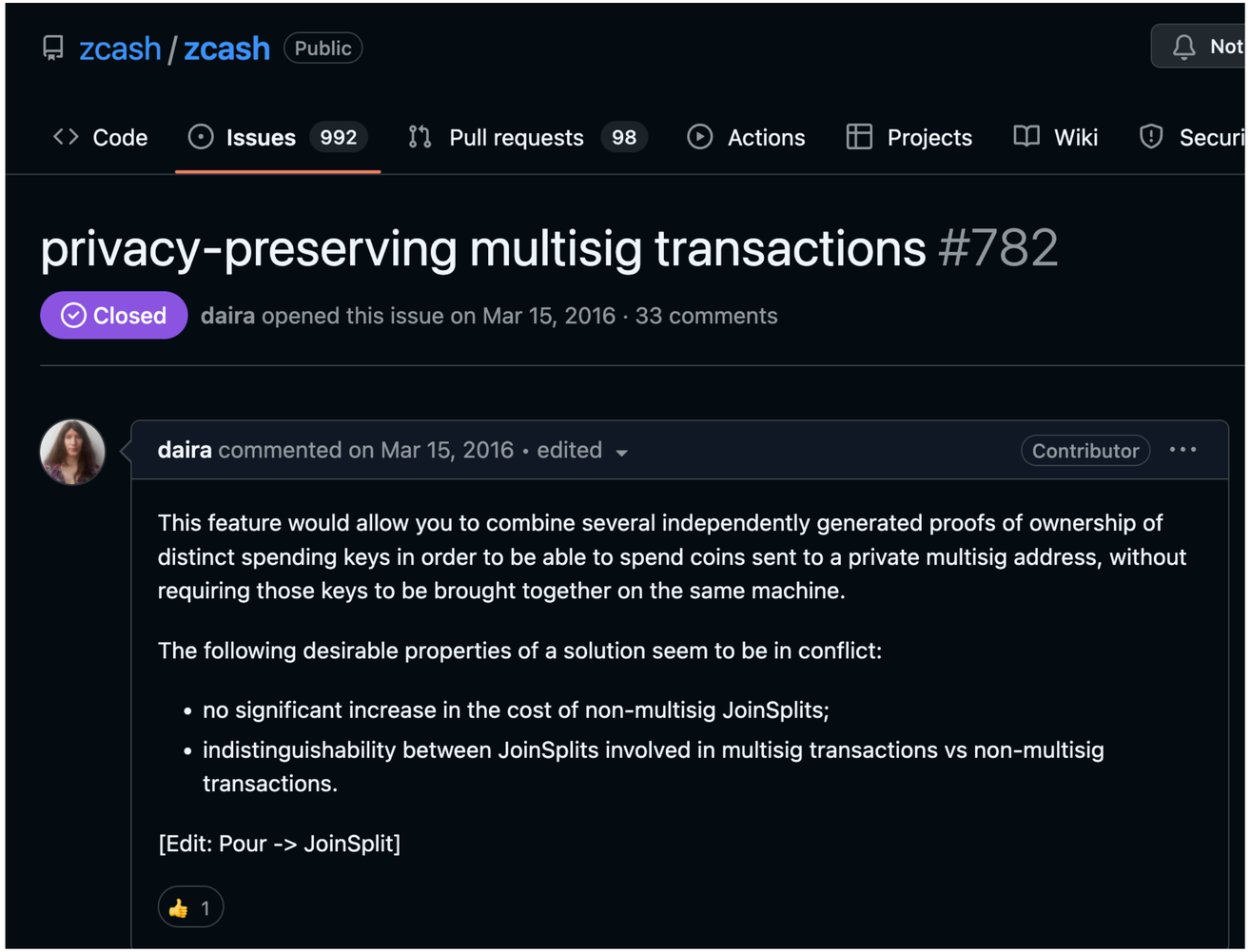
How to share sk ?

$$z \leftarrow r + c \cdot sk$$

$$sig = (R, z)$$

Motivation

Zcash Motivation



Existing Work

5-Round

Provably Secure Distributed Schnorr Signatures and a (t, n) Threshold Scheme for Implicit Certificates

Douglas R. Stinson[1,2] and Reto Strobl[3]

3-Round

Simple Schnorr Multi-Signatures with Applications to Bitcoin

Gregory Maxwell, Andrew Poelstra[1], Yannick Seurin[2], and Pieter Wuille[1]

Research Question: Could we design a round-efficient threshold Schnorr scheme?

A First Attempt

PK_1

$$r_1 \leftarrow \mathbb{Z}_q$$

$$R_1 = g^{r_1}$$

$$R = \prod_{i=1}^{n} R_i$$

$$c = H(PK, R, m)$$

$$z_1 = r_1 + csk_1\lambda_1$$

R_1 →

← $\{R_2 \ldots, R_n\}$

↓

$(z_1, \ldots, z_n)$

PK_2

PK_n

Combine/Verify

$$z = \Sigma_{i=1}^{n} z_i$$

$$sig = (R, z)$$

$$c \leftarrow H(PK, m, R)$$

$$R \cdot PK^c = g^z$$

✓

Secure in a sequential setting

Concurrent Security: ROS

Session 1

sk_1

$R_1^{(1)}$

$R_2^{(1)}$

$\cdots$

Session k

$R_1^{(k)}$

$R_2^{(k)}$

sk_2

Can forge!

(Random inhomogeneities in a Overdetermined Solvable system of linear equations)

[DEFKLNS19, BLLOR21]

Affected:

- multi-signatures
- threshold signatures
- blind signatures

One-More Forgery Adversary

PK_1

$$R_1 = g^{r_1}$$

$$R_1' = g^{r_1'}$$

$$R_1'' = g^{r_1''}$$

$$R = \prod_{i=1}^{n} R_i$$

$$c = H(PK, R, m)$$

$$z_1 = r_1' + csk_1\lambda_1$$

R_1 →

R_1' →

R_1'' →

← $\{R_1', R_2 \ldots, R_n\}$

PK_2

PK_n

Challenge to Adversary: Given ℓ signing sessions, find:

1. A challenge $c^* = H(R^*, m^*)$
2. Such that $c^* = \Sigma_{i=1}^{\ell} c_i$
3. Where $c_i = H(R_i, m_i)$ (from a valid signing session)
4. Such that

$$z^* = \sum_{i=1}^{\ell} z_i = \sum_{i=1}^{\ell} r_i + c_i\lambda_i sk_i = \sum_{i=1}^{\ell} r_i + c^* sk$$

Requires the adversary to adaptively choose its contributions after seeing the victim's

Prior Insecure Schemes

Scheme	KVf	KAg	Sign	Vf	Rounds	pk domain	Signature domain	PK domain	Security
BCJ1 [BCJ08]	$1\mathbb{G}^2$		$1\mathbb{G}^2 + 1\mathbb{G}^3$	$3\mathbb{G}^2$	2	$\mathbb{G} \times \mathbb{Z}_q^2$	$\mathbb{G}^2 \times \mathbb{Z}_q^3$	$\mathbb{G}$	Insecure
BCJ2 [BCJ08]			$1\mathbb{G} + 2\mathbb{G}^2$	$1\mathbb{G}^{n+1} + 2\mathbb{G}^2$	2	$\mathbb{G}$	$\mathbb{G}^3 \times \mathbb{Z}_q^3$	$\mathbb{G}$	Insecure
MWLD [MWLD10]			$1\mathbb{G}^2$	$1\mathbb{G}^{n+2}$	2	$\mathbb{G}$	$\mathbb{Z}_q^3$	$\mathbb{G}^n$	Insecure
CoSi [STV+16]	$1\mathbb{G}^2$		$1\mathbb{G}$	$1\mathbb{G}^2$	2	$\mathbb{G} \times \mathbb{Z}_q^2$	$\mathbb{Z}_q^2$	$\mathbb{G}$	Insecure
MuSig [MPSW18a]		$1\mathbb{G}^n$	$1\mathbb{G}$	$1\mathbb{G}^2$	2	$\mathbb{G}$	$\mathbb{G} \times \mathbb{Z}_q$	$\mathbb{G}$	Insecure
mBCJ (this work)	$1\mathbb{G}^2$		$1\mathbb{G}^2 + 1\mathbb{G}^3$	$3\mathbb{G}^2$	2	$\mathbb{G} \times \mathbb{Z}_q^2$	$\mathbb{G}^2 \times \mathbb{Z}_q^3$	$\mathbb{G}$	DL, ROM
BN [BN06]			$1\mathbb{G}$	$1\mathbb{G}^{n+1}$	3	$\mathbb{G}$	$\mathbb{G} \times \mathbb{Z}_q$	$\mathbb{G}^n$	DL, ROM
BDN-MSDL [BDN18,MPSW18b]		$1\mathbb{G}^n$	$1\mathbb{G}$	$1\mathbb{G}^2$	3	$\mathbb{G}$	$\mathbb{G} \times \mathbb{Z}_q$	$\mathbb{G}$	DL, ROM
B-Pop [Bol03,RY07]	$2P$		$1\mathbb{G}_1$	$2P$	1	$\mathbb{G}_1 \times \mathbb{G}_2$	$\mathbb{G}_1$	$\mathbb{G}_2$	co-CDH, ROM
WM-Pop [LOS+06,RY07]	$2P$		$1\mathbb{G}_1 + 1\mathbb{G}_2$	$2P$	1	$\mathbb{G}_1 \times \mathbb{G}_2 \times \mathbb{G}_T$	$\mathbb{G}_1 \times \mathbb{G}_2$	$\mathbb{G}_T$	co-CDH
BDN-MSP [BDN18]		$1\mathbb{G}_2^n$	$1\mathbb{G}_1$	$2P$	1	$\mathbb{G}_2$	$\mathbb{G}_1$	$\mathbb{G}_2$	co-CDH, ROM

[DEFKLNS19]

Also Musig1, FROST v1 !

Three Round Fix

Three Round Fix: Force adversary to commit to its contributions.

MuSig(1)

[MPSW18, MPSW19, C**K**M23, M23]

PK_1

$r_1 \leftarrow \mathbb{Z}_q$

$R_1 = g^{r_1}$

$cm_i = H(R_i, m, S = \{1,\ldots,t\})$

$R = \prod_{i=1}^{t} R_i$

$c = H(PK, R, m)$

$z_1 = r_1 + csk_1$

cm_1

$\{cm_1, \ldots, cm_t\}$

R_1

$\{R_1 \ldots, R_t\}$

$(z_1, \ldots, z_t)$

PK_2

PK_t

Combine/Verify

$z = \Sigma_{i=1}^{t} z_i$

$sig = (R, z)$

$c \leftarrow H(PK, m, R)$

$R \cdot PK^c = g^z$

✔

Three *online* rounds

Similar Techniques: Sparkle, ZeroS Lindell22

Solution #2: “Tweak” the signature.

FROST

Sharing sk (threshold, trusted dealer)

Step 1: Sample $(sk, \{a_1, \ldots, a_t\}) \leftarrow \mathbb{Z}_q^t$

Step 2: Define $f(x) = sk + \Sigma_{j=1}^{t} a_j x^j$

Shamir secret sharing

Step 3: Define $sk_j = f(j), \forall j \in \{1, \ldots, n\}$

Step 4: Define $PK = g^{sk}$

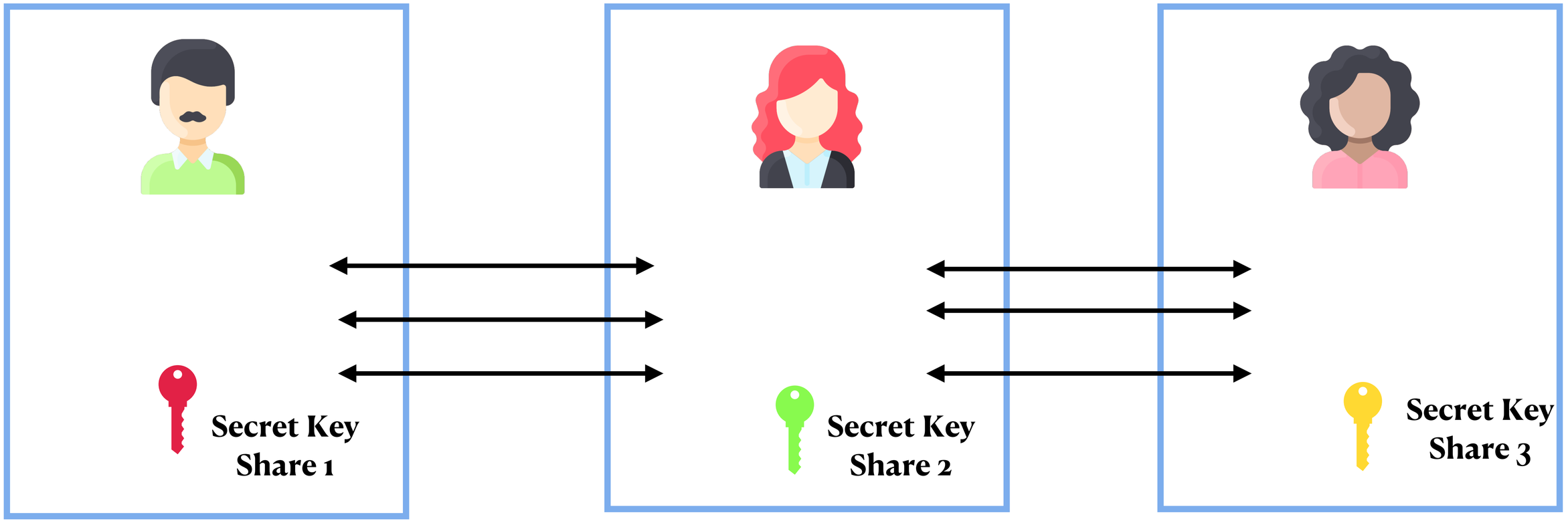
$((i, sk_i), PK)$

Recovering sk

$$sk \leftarrow \sum_{i \in S} sk_i \lambda_i,\ S \subseteq \{1, \ldots, n\}, |S| \geq t + 1$$

- λ_i is the ith Lagrange coefficient; determined using only (i, S)
- Intuition: $t + 1$ points uniquely define a polynomial of degree t

Distributed Key Generation



FROST

[**K**G20, C**K**M21, BC**K**MTZ22]

- **F**lexible **R**ound-**O**ptimized **S**chnorr **T**hreshold **S**ignatures [Komlo & Goldberg, 2020]
  1. PedPop: Distributed Key Generation (DKG).
  2. Two-round threshold signing that is concurrently secure .
- Designed to solve needs in the Zcash ecosystem; now adopted as an industry standard

eprint.iacr.org/2020/852

FROST

[**K**G20, C**K**M21, BC**K**MTZ22, CGRS23]

MuSig2 signing Is similar! More on this later

PK_1

$$r_1, s_1 \leftarrow \mathbb{Z}_q$$

$$R_1 = g^{r_1}; S_1 = g^{s_1}$$

$$a = H(PK, m, \{(i, R_i, S_i)\}_{i=1}^{t})$$

$$R = \prod_{i=1}^{t} R_i S_i^a$$

$$c = H(PK, R, m)$$

$$z_1 = r_1 + s_1 a + c s k_1 \lambda_1$$

(R_1, S_1)

$\{(R_1, S_1) \ldots, (R_t, S_t)\}$

$(z_1, \ldots, z_t)$

PK_2

PK_t

Combine/Verify

$$z = \Sigma_{i=1}^{t} z_i$$

$$sig = (R, z)$$

$$c \leftarrow H(PK, m, R)$$

$$R \cdot PK^c = g^z$$

First Round:

- Can be batched!
- Agnostic to message & signing set

FROST: Correctness

$$z = \sum_{i=1}^{t} z_i$$

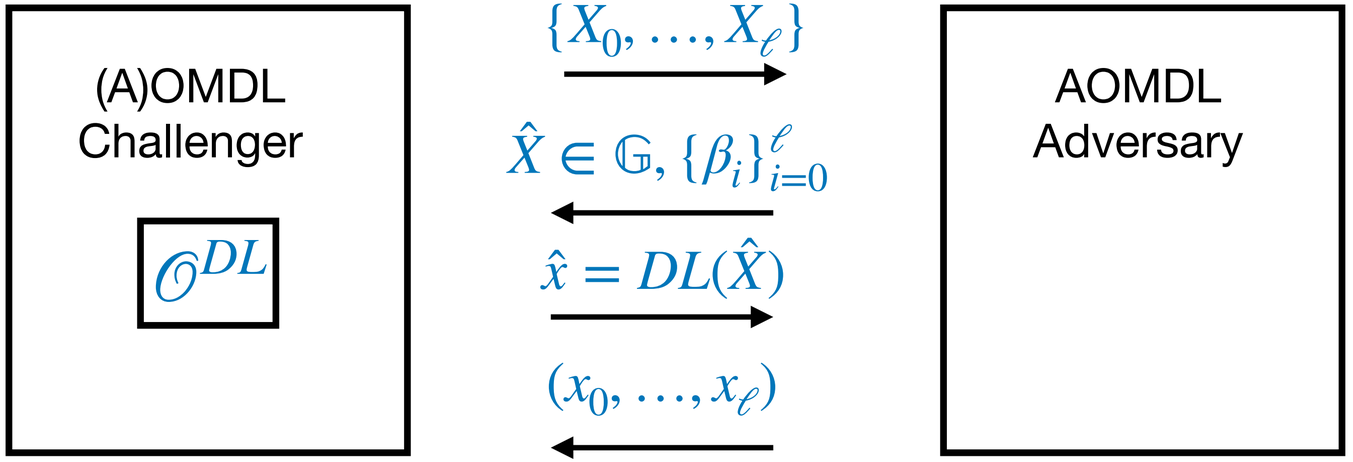
$$= \sum_{i=1}^{t} r_i + \sum_{i=1}^{t} c \cdot sk_i \cdot \lambda_i$$

$$= \sum_{i=1}^{t} r_i + csk$$

By Shamir secret sharing

Proving Security

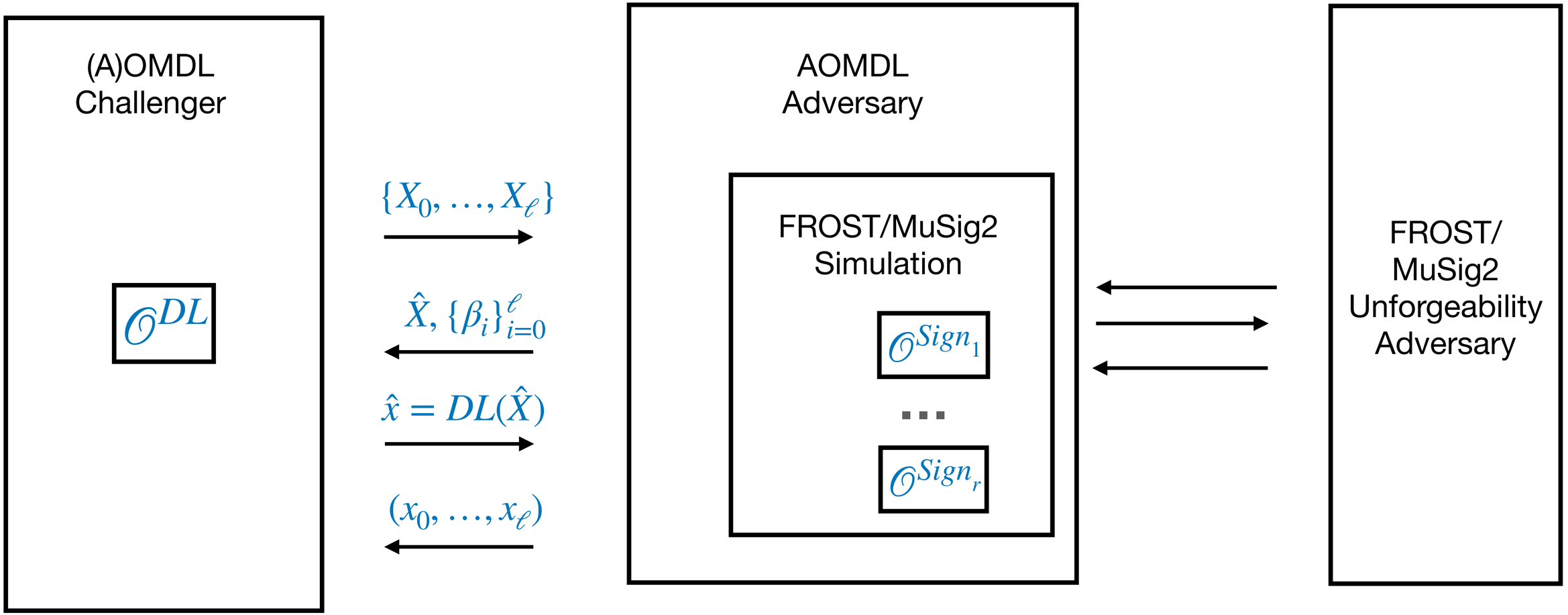
- Goal: Show that if a mathematical assumption is hard, then a scheme is secure.
- Contrapositive: If an adversary can break the security of a scheme, possible to efficiently solve a hard mathematical problem.
- AOMDL Assumption: Algebraic One-More Discrete Logarithm Assumption.
- Given $\ell + 1$ discrete logarithm challenges $\{X_0, \ldots, X_\ell\} \in \mathbb{G}$, hard to output $\ell + 1$ solutions $\{x_0 \ldots, x_\ell\} \in \mathbb{Z}_q$ with only ℓ queries to a DL solution oracle.



Static Security of FROST/MuSig2

[NRS21, BC**K**MTZ22]

Concurrently secure under (A)OMDL in the Random Oracle Model (ROM).



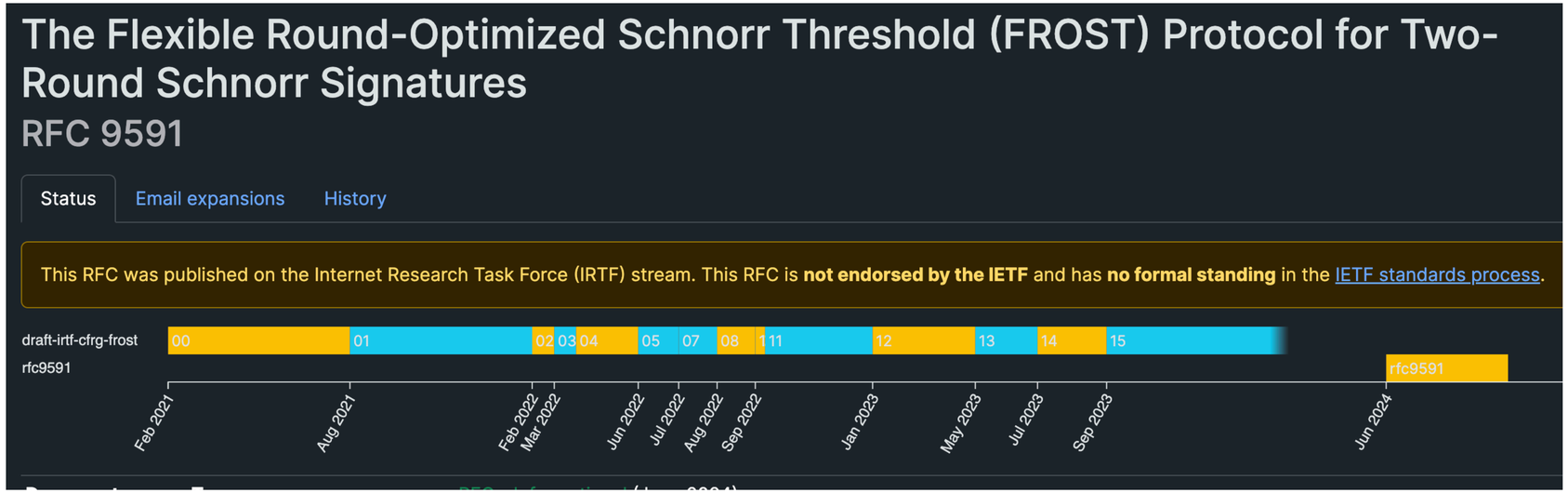
	Scheme	Assumptions	Signing Rounds
Multi-sigs (n-of-n)	**MuSig** [MPSW18, BDN18] **SimpleMuSig** [BDN18, CKM21]	DL+ROM	3
	MuSig2 [NRS21] **DWMS** [AB21] **SpeedyMuSig** [CKM21]	(A)OMDL+ROM	2
Threshold (t-of-n)	**Lindell22** **Sparkle** [CKM23]	Schnorr DL+ROM	3
	FROST [KG20, BCKMTZ22] **FROST2** [CKM21]	(A)OMDL+ROM	2

Algebraic One-More Discrete Log (AOMDL):

\- stronger assumption

\+ partially non-interactive schemes

Standardization




```
ZIP: 312
Title: FROST for Spend Authorization Signatures
Owners: Conrado Gouvea <conrado@zfnd.org>
        Chelsea Komlo <ckomlo@uwaterloo.ca>
        Deirdre Connolly <deirdre@zfnd.org>
Status: Draft
Category: Wallet
Created: 2022-08-dd
License: MIT
Discussions-To: <https://github.com/zcash/zips/issues/382>
Pull-Request: <https://github.com/zcash/zips/pull/662>
```

FROST NIST Submission Team

Elizabeth Crites

Conrado Gouvea

Ian Goldberg

Jack Grigg

Jonathan Katz

Chelsea Komlo

Mary Maller

Stefano Tessaro

Nikita Sorokovikov

Denis Varlakov

Chenzhi Zhu

NISTIR 8214C (Draft)

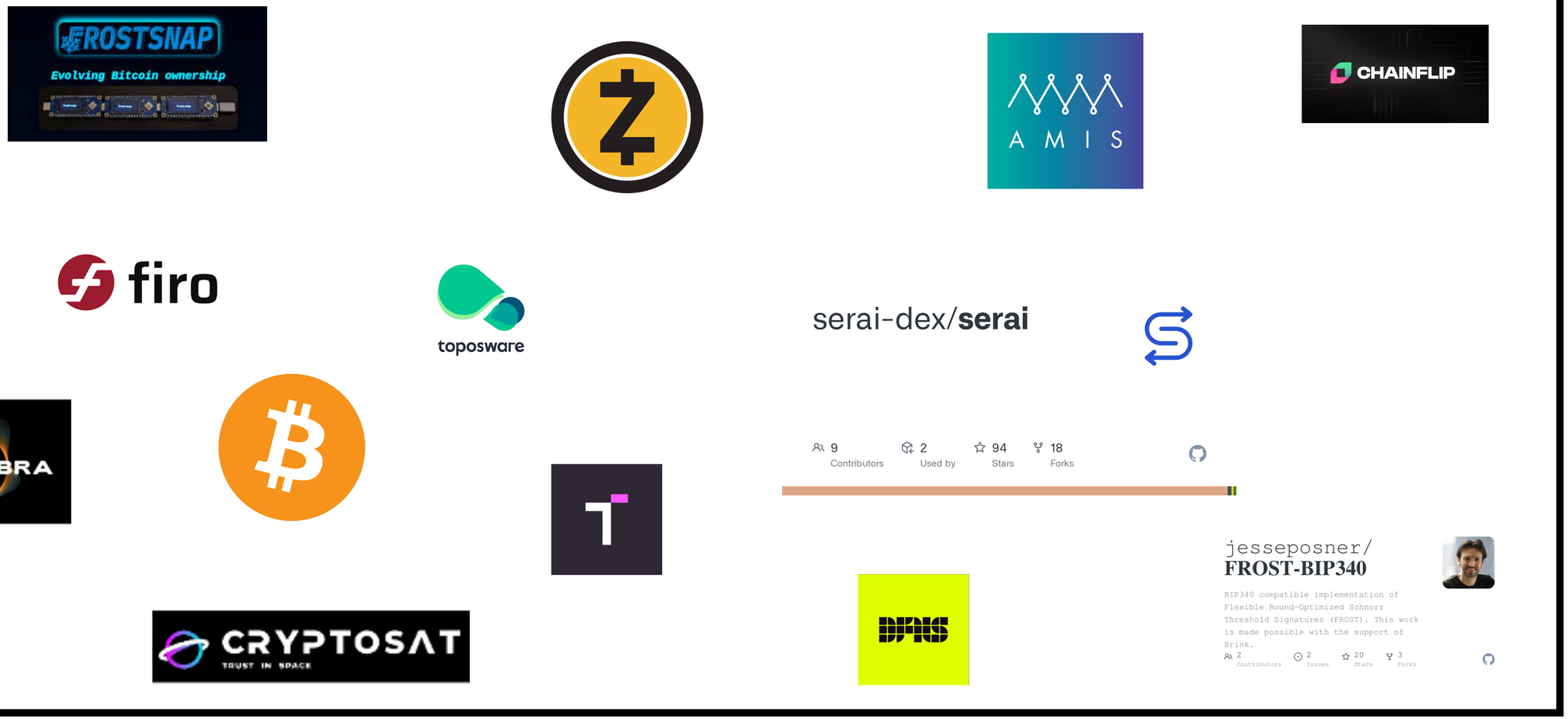
NIST First Call for Multi-Party Threshold Schemes

Date Published: January 25, 2023
Comments Due: April 10, 2023
Email Comments to: nistir-8214C-comments@nist.gov

Author(s)

Luís T. A. N. Brandão (Strativia), Rene Peralta (NIST)

FROST in Practice



Subsequent Work

How to Prove Schnorr Assum
Security of Multi- and Thres

Elizabeth Crites[1], Chelsea Komlo[2], and Mar

Robust FROST

ROAST: Robust Asynchronous Schnorr Threshold Signatures

Tim Ruffing
Blockstream
crypto@timruffing.de

Viktoria Ronge
Friedrich-Alexander-Universität
Erlangen-Nürnberg
ronge@cs.fau.de

Elliott Jin
Blockstream
eyj@blockstream.com

Jonas Schneider-Bensch
CISPA Helmholtz Center for
Information Security
jonas.schneider-bensch@cispa.de

Dominique Schröder
Friedrich-Alexander-Universität
Erlangen-Nürnberg
dominique.schroeder@fau.de

Better than Advertised Security for Non-interactive Threshold Signatures

Mihir Bellare[1], Elizabeth Crites[2], Chelsea Komlo[3], Mary Maller[4], Stefano Tessaro[5], and Chenzhi Zhu[5(✉)]

Improved proofs of security for FROST

any faces of Schnorr

Victor Shoup
Offchain Labs

April 8, 2024

A Formal Treatment of Distributed Key Generation, and New Constructions

Chelsea Komlo, Ian Goldberg, Douglas Stebila

Threshold Key Generation

Threshold and Multi-Signature Schemes from Linear Hash Functions*

Stefano Tessaro and Chenzhi Zhu

Paul G. Allen School of Computer Science & Engineering
University of Washington, Seattle, US
{tessaro,zhucz20}@cs.washington.edu

Glacius: Threshold Schnorr Signatures from DDH with Full Adaptive Security

Renas Bacho[1,2], Sourav Das[3], Julian Loss[1] and Ling Ren[3]

[1]CISPA Helmholtz Center for Information Security, Saarbrücken, Germany
[2]Saarland University, Saarbrücken, Germany
[3]University of Illinois at Urbana Champaign
{renas.bacho, loss}@cispa.de, {souravd2, renling}@illinois.edu

Fully Adaptive Schnorr Threshold Signatures

Elizabeth Crites[1], Chelsea Komlo[2], and Mary Maller[3]

Practical Schnorr Threshold Signatures Without the Algebraic Group Model

Hien Chu[1], Paul Gerhart[1], Tim Ruffi

[1] Friedrich-Alexander-U
[2] Blockst

Post Quantum (Lattice)
MuSig2 and FROST

Threshold
schemes with weaker
security assumptions

hreshold Signature from e-More Learning with Errors

Thomas Espitau[1], Shuichi Katsumata[1,2], Kaoru Takemure* [1,2]

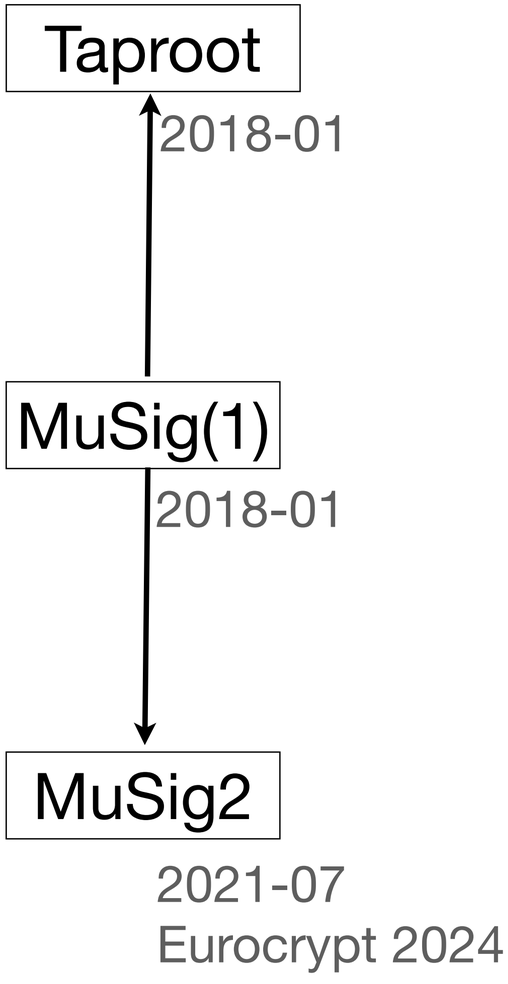
[1]PQShield
{thomas.espitau, shuichi.katsumata, kaoru.takemure}@pqshield.com
[2]AIST

MuSig-L: Lattice-Based Multi-Signature With Single-Round Online Phase*

Cecilia Boschini[1], Akira Takahashi[2], and Mehdi Tibouchi[3]

Towards MuSig2

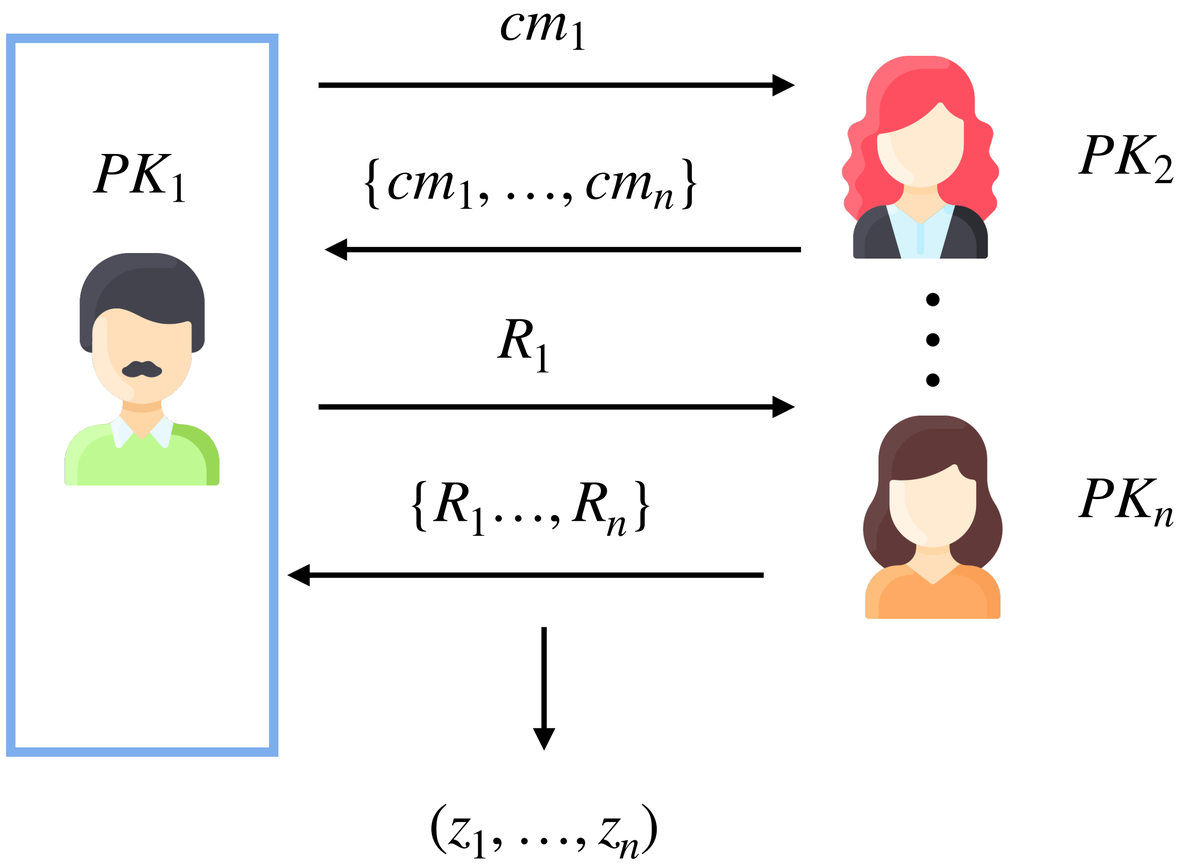
MuSig & Bitcoin



MuSig(1) [MPSW19]

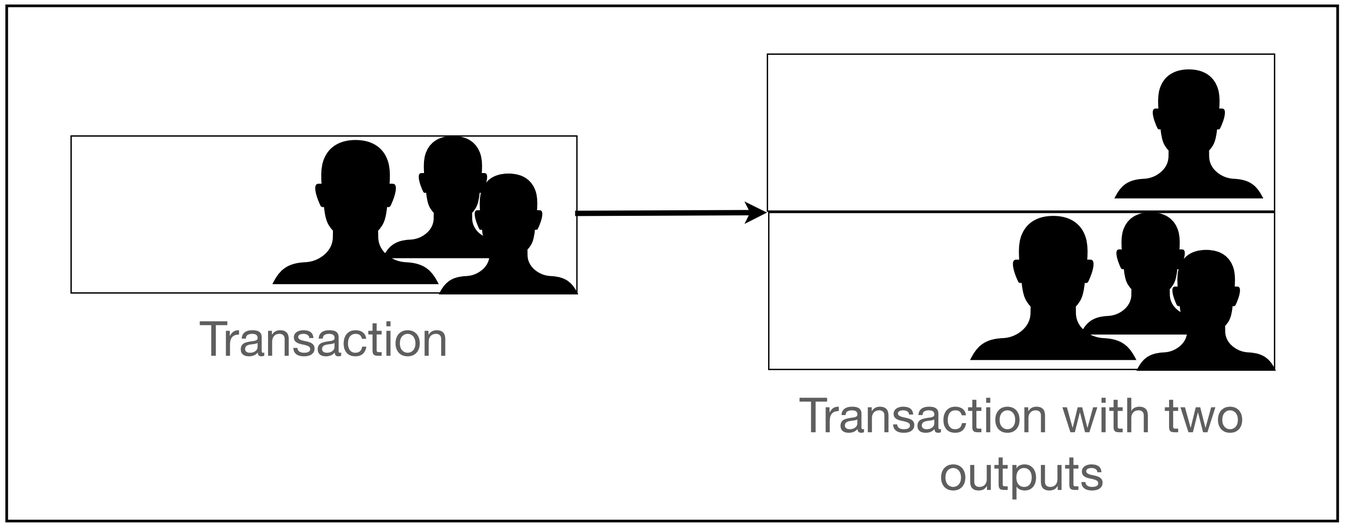
Simple Schnorr Multi-Signatures with Applications to Bitcoin

- First **Schnorr signature-compatible** multi-signature scheme with non-interactive, **public key aggregation**.
  - $\widetilde{PK} = \mathsf{KeyAgg}(\{PK_1, \ldots, PK_n\})$
- "**[Bitcoin] moving to Schnorr signatures** would definitely help deploying compact multi-signatures"



MuSig(1) Motivation for Bitcoin

- Bitcoin has always supported "trivial multi-signatures"
  - $PK = (PK_1, \ldots, PK_n)$, $sig = (sig_1, \ldots, sig_n)$
- Using MuSig allows
  - Improving **privacy**
  - Saving transaction **fees**



Trivial Multisignatures have terrible privacy: Output 2 is clearly the change output.

Taproot Soft-Fork

- **BIP 340**: Schnorr Signatures
  - 64 bytes, over curve secp256k1
  - designed for multi- and threshold-signatures
- **BIP 341**: Taproot spending rules

Taproot Commitment

$\text{commit} : \mathbb{M} \times \mathbb{G} \to \mathbb{G}$

Key feature: Knowledge of the discrete log x of the input group element allows computing the DL of the the output group element:

$$\text{taptweak} : \mathbb{M} \times \mathbb{Z}_p \to \mathbb{Z}_p$$

$$\text{DL}_g(\text{commit}(m, g^x)) = \text{taptweak}(m, x)$$

Taproot spending rules: A coin containing Taproot commitment can be **spent in two ways**:

1. **Key spend:** requires a Schnorr signature using the commitment as the public key. The secret key is computed via taptweak.

2. **Script spend:** requires opening the commitment to m, which is interpreted as a tree of alternative spending conditions, and satisfying one of the conditions.

Taproot Assumption

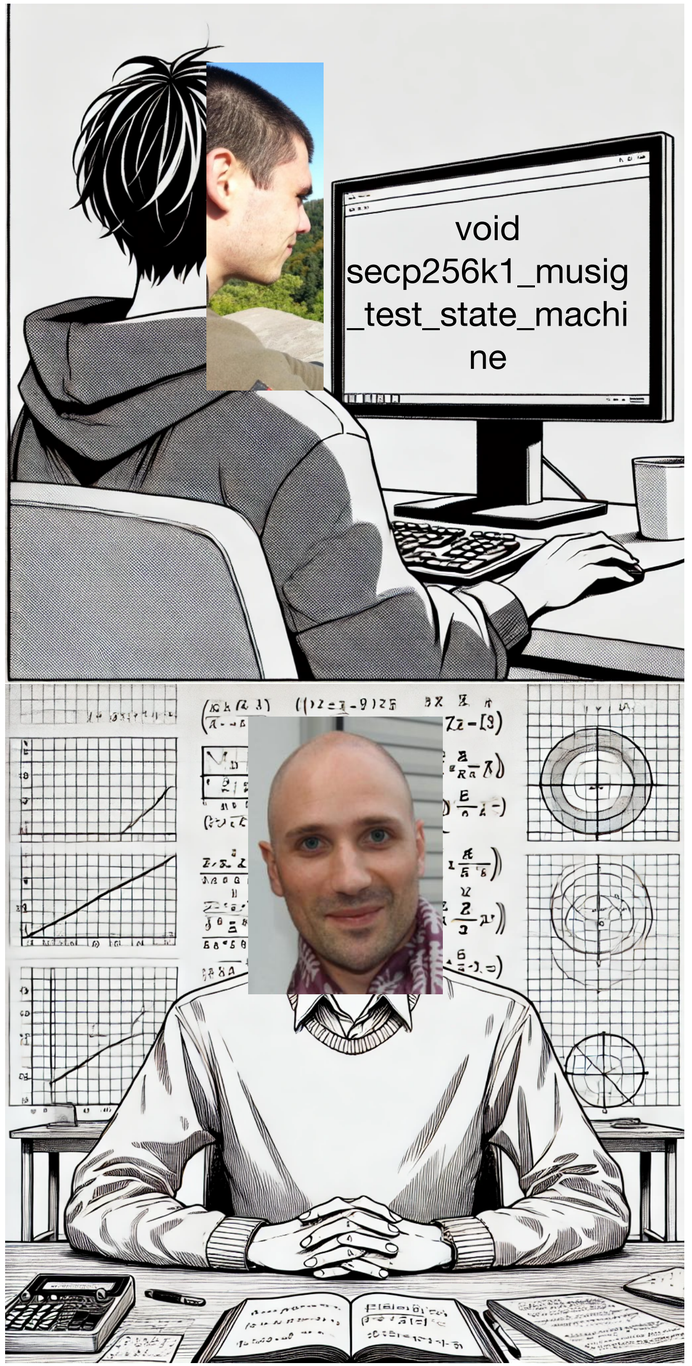
> "[..] it is almost always the case that interesting scripts have a logical top level branch which allows satisfaction of the contract with nothing other than a **signature by all parties**."

Greg Maxwell in the original Taproot post on the Bitcoin mailing list

So, if all participants of some contract agree, they can use the **key spend** path with a MuSig multi-signature.

This does not require revealing the alternative spending conditions and is **indistinguishable** from a normal single-signature transaction.

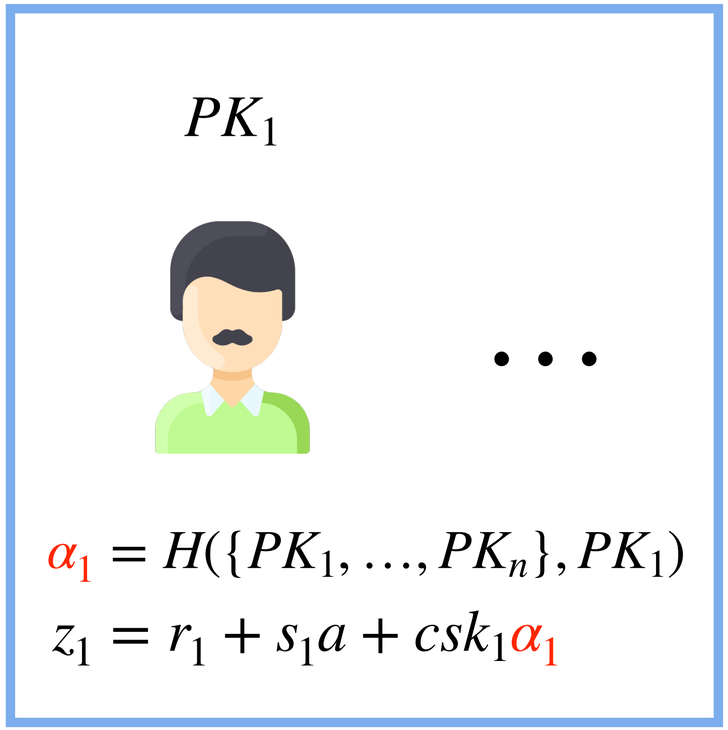
MuSig2 Origin



MuSig2 [NRS21]

Simple Two-Round Schnorr Multi-Signatures

$$\widetilde{PK} = \mathsf{KeyAgg}(\{PK_1, \ldots, PK_n\}) = \prod_{i=1}^{n} PK_i^{\alpha_i}$$

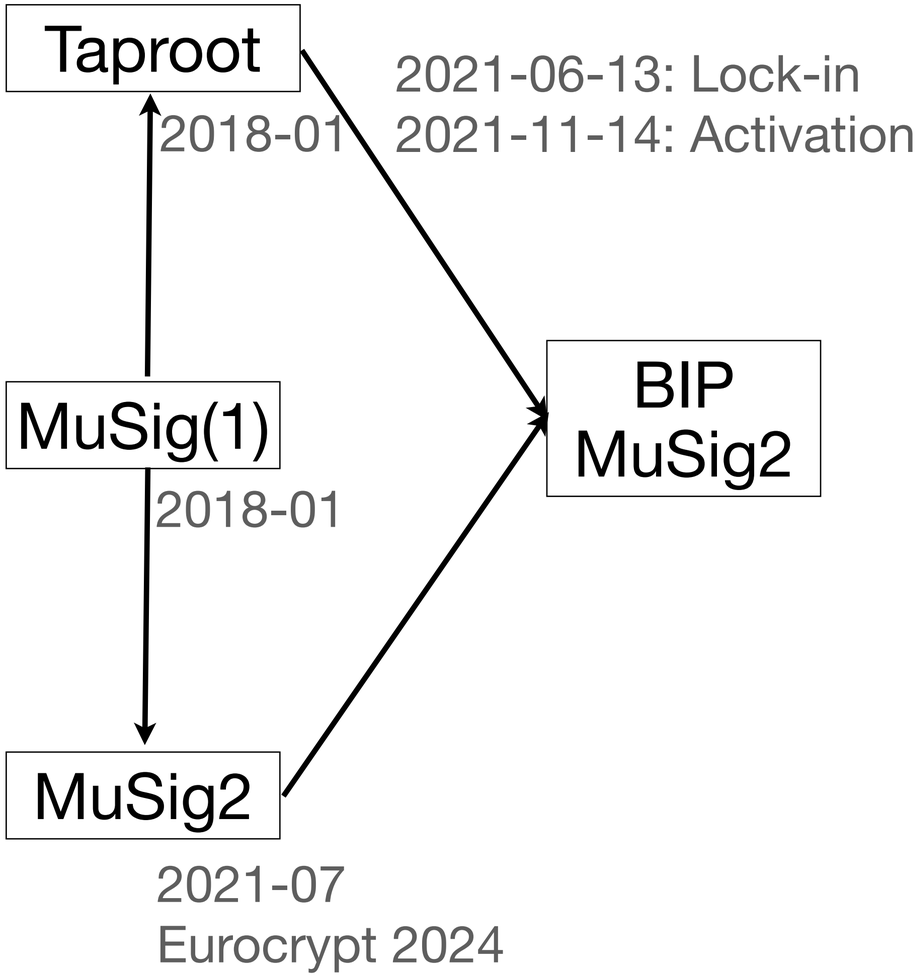


First Round:

- Can be batched!
- Agnostic to message & signing set

MuSig2 in Practice

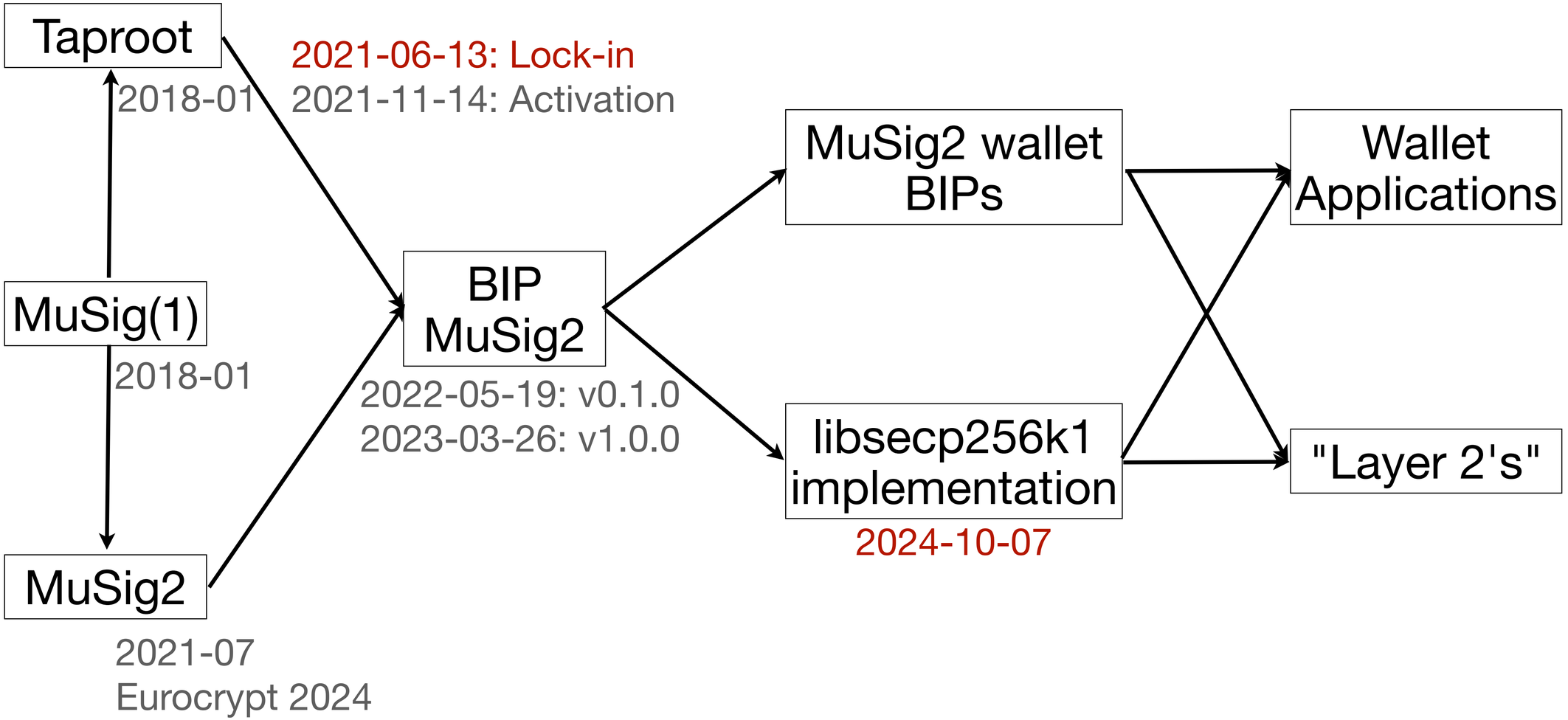
MuSig & Bitcoin



BIP MuSig2

- **Specification**, compatible with BIP 340.
- Includes:
  - Python **reference implementation**
  - **exhaustive test vectors** (that test all kinds of failure conditions)
- **Supports Taproot commitments** and a few other other features not part of the MuSig2 paper.

MuSig & Bitcoin



Can we accelerate the development timeline?

While BIP development and implementation occurred **in parallel**, several factors slowed down the timeline:

- Need to get **feedback from experts**; experts tend to be busy with other things
- Need to get **feedback from potential users**
  - **Implementations take time** and usually they don't happen until BIP/library is finalized
  - Limited urgency due to **low network fee pressure**
- Maximize **clarity** of BIP
  - Build exhaustive **test framework**
  - Validate features of the BIP that are **not covered by MuSig2 security proof**
- BIP & Library: Design **misuse-resistant API**

Can we accelerate the development timeline?

Theory vs. Practice: Related Keys

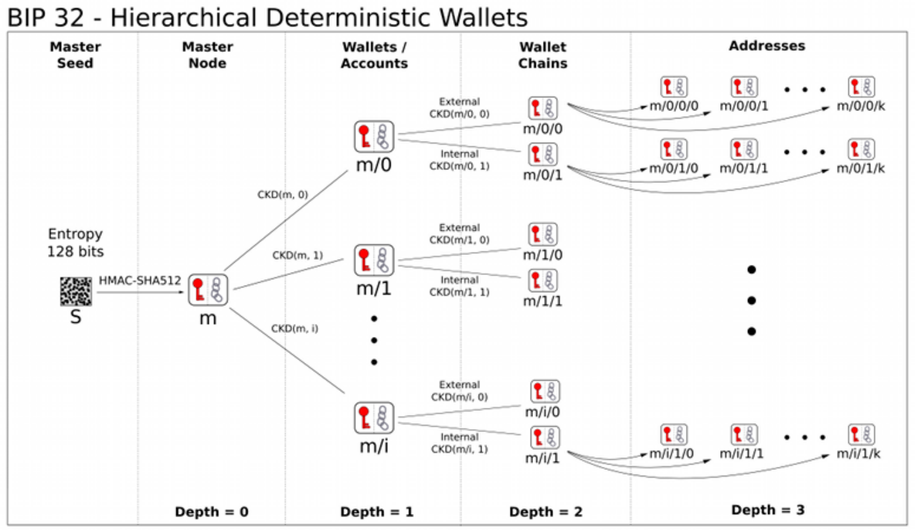
KeyGen()

$x \leftarrow\$ \mathbb{Z}_p \; ; \;\; X := g^x$

$sk := x \; ; \;\; pk := X$

return (sk, pk)

Key generation in MuSig2 paper



Key generation in practice. Keys are derived from master key and therefore related.

```
It is still true that cryptography is hard, unfortunately. Yannick Seurin, Tim
Ruffing, Elliott Jin, and I discovered an attack against the latest version of
BIP MuSig2 in the case that a signer's individual key A = a*G is tweaked before
giving it as input to key aggregation.
```

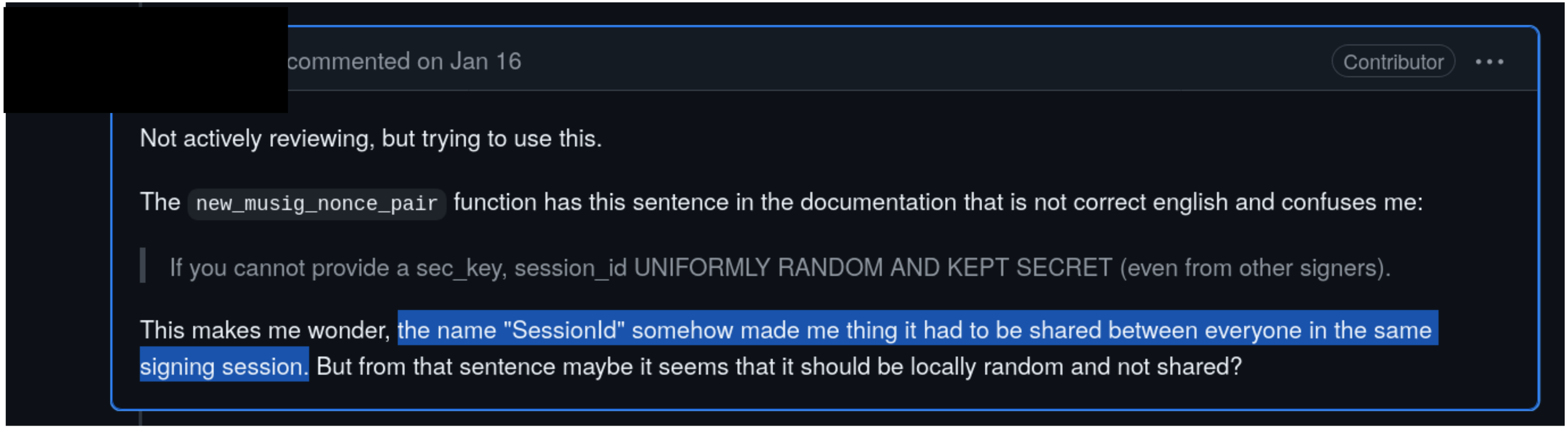
Can we accelerate the development timeline?

Theory vs. Practice: Related Keys

- Attack requires **specific circumstances** and active exploitation
  - Security impact: **Moderate**
- **fix is straightforward,** happened before 1.0.0
- **Discovery was incidental** while working on "nested" MuSig
- **Key Learning:** Security proofs should should cover practical use as much as possible
  - and in particular, **cover related keys** (aka tweaked keys)

Can we accelerate the development timeline?

Misuse-resistant API Design is Important



We then renamed SessionID to SessionSecRand

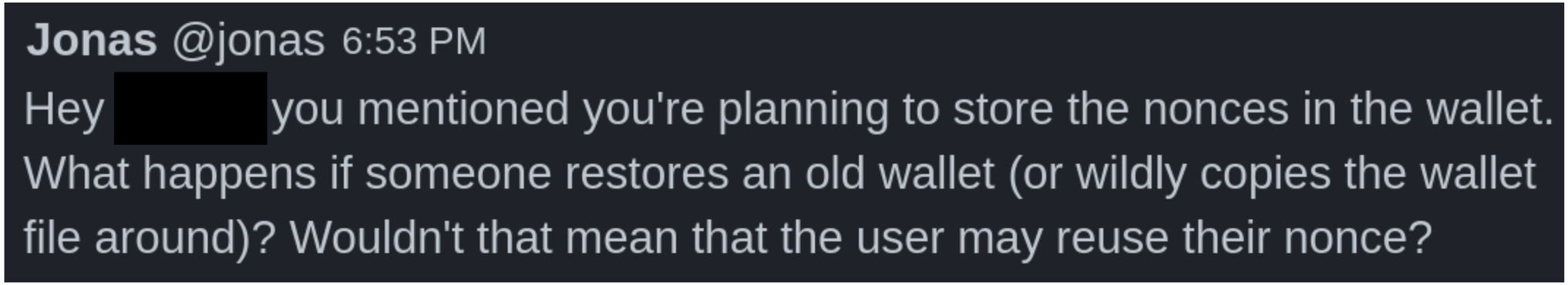
```
 *  WARNING: This structure MUST NOT be copied or read or written to directly. A
 *  signer who is online throughout the whole process and can keep this
 *  structure in memory can use the provided API functions for a safe standard
 *  workflow.
 *
 *  Copying this data structure can result in nonce reuse which will leak the
 *  secret signing key.
 */
typedef struct secp256k1_musig_secnonce {
```

```
*  3. Avoid copying (or serializing) the secnonce. This reduces the possibility
*     that it is used more than once for signing.
```

2. The `secp256k1_musig_secnonce` structure is never copied or serialized. See also the comment on `secp256k1_musig_secnonce` in `include/secp256k1_musig.h`.

Can we accelerate the development timeline?

Misuse-resistant API Design is Important



- Turned out, yes, this could have resulted **nonce reuse**.
- While the developer was **aware of the risks** with nonce reuse...
- They failed to consider that **this sort of copying would lead to reuse** because "generating a new random nonce each time should be good enough".
- What can we do about this?
  - Design crypto schemes that don't have such problems?

MuSig2 Applications

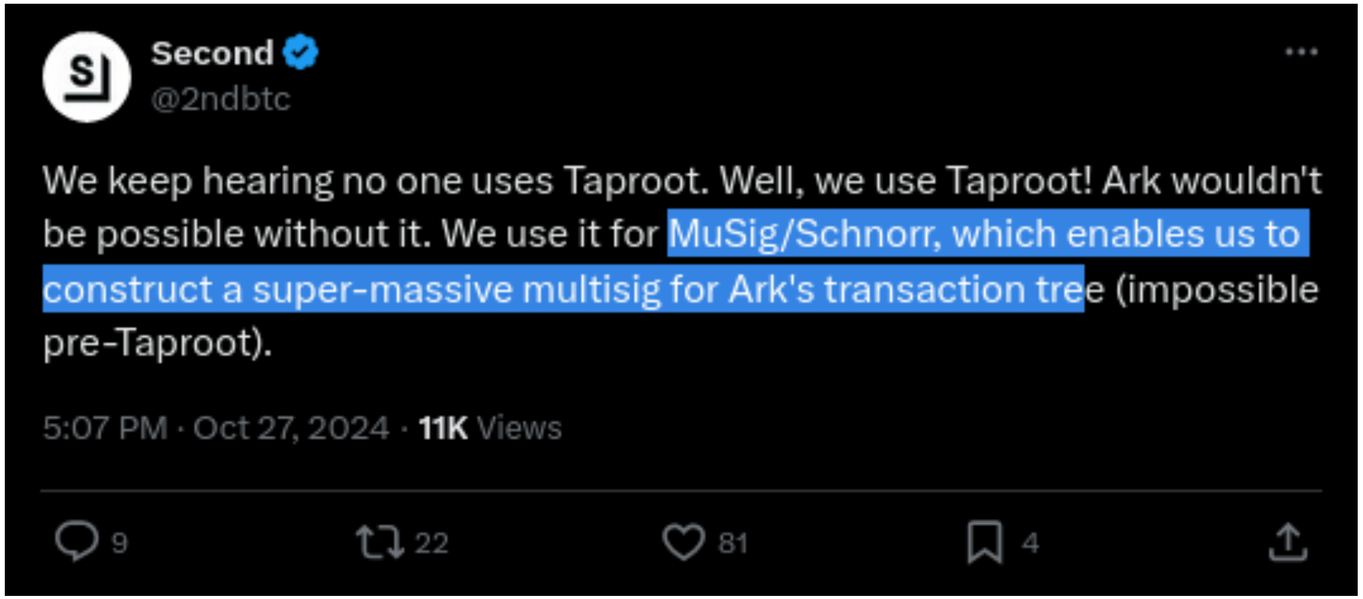
Applications

	Application	
In Production*	Wallet with third-party programmable cosigner	2-of-3 (using Taproot tree)
	Swap between Lightning Network and on-chain.	2-of-2
In Development	Lightning Network channels	2-of-2
	Bitcoin Core wallet	flexible
	...	...

* Adoption of MuSig2 is **impossible to observe** on the blockchain

More Applications

- **Covenant-less Ark (clArk) protocol**:



  - would be prohibitively expensive with trivial multisignatures
- **Mercury Layer:** refresh MuSig2 secret keys to transfer coins without a transaction using a (semi-) trusted server; impossible with traditional multisig

Most Requested MuSig2/FROST Extensions

- (sequential) **blind** FROST/MuSig2 signing
- **nested** (AKA recursive) FROST/MuSig2
  - each of the aggregated keys can itself be a MuSig2 aggregate or FROST key
- **stateless** MuSig2/FROST
  - exists (e.g., [NRSW20, KG24]), but very complex or requires honest majority
- Combine MuSig2/FROST with **Silent Payments**
- **Dynamic membership** MuSig2/FROST
  - change n, t, or members of the signing group without a Bitcoin transaction

Learnings

- Research is valuable when **informed by real-world practice**. **Communication** is key.
- Approach problems with your **unique perspective** and **persist** in finding solutions.
- Cryptographic schemes should **take practical concerns into account** (statefulness, related-keys, misuse-resistance, communication rounds,...).
- Academic findings need **active effort** to transition into practice.
- **Thank you** to everyone who helped move FROST and MuSig2 to practice!

MuSig2

Security proofs

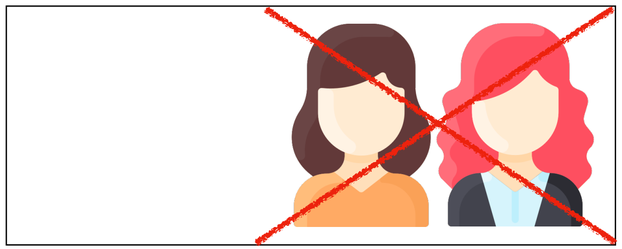
Model	Number of Nonces
ROM	4
ROM + AGM	2

- ROM + Algebraic Group Model (AGM) is a stronger model. No (serious) scheme proven secure in the AGM has been broken.
- 4 nonces **artifact** of ROM-only proof technique.

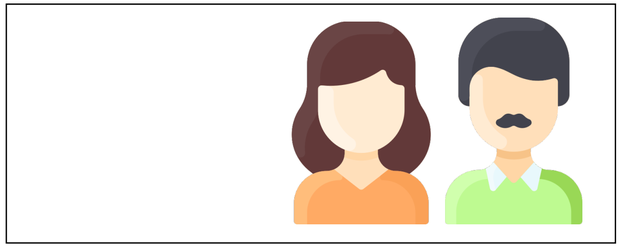
More Applications

Mercury Layer

Motivation: key refresh to transfer coins without a transaction using a (semi-) trusted server; impossible with traditional multisig



Transaction



Same transaction but with reshared transaction output