

# 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 Seurin

Ledger

Why a shared prize between  
FROST and MuSig2?

# A Story in Four Parts

 Cryptology ePrint Archive Paper

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

TR musig2-...7.pdf

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

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

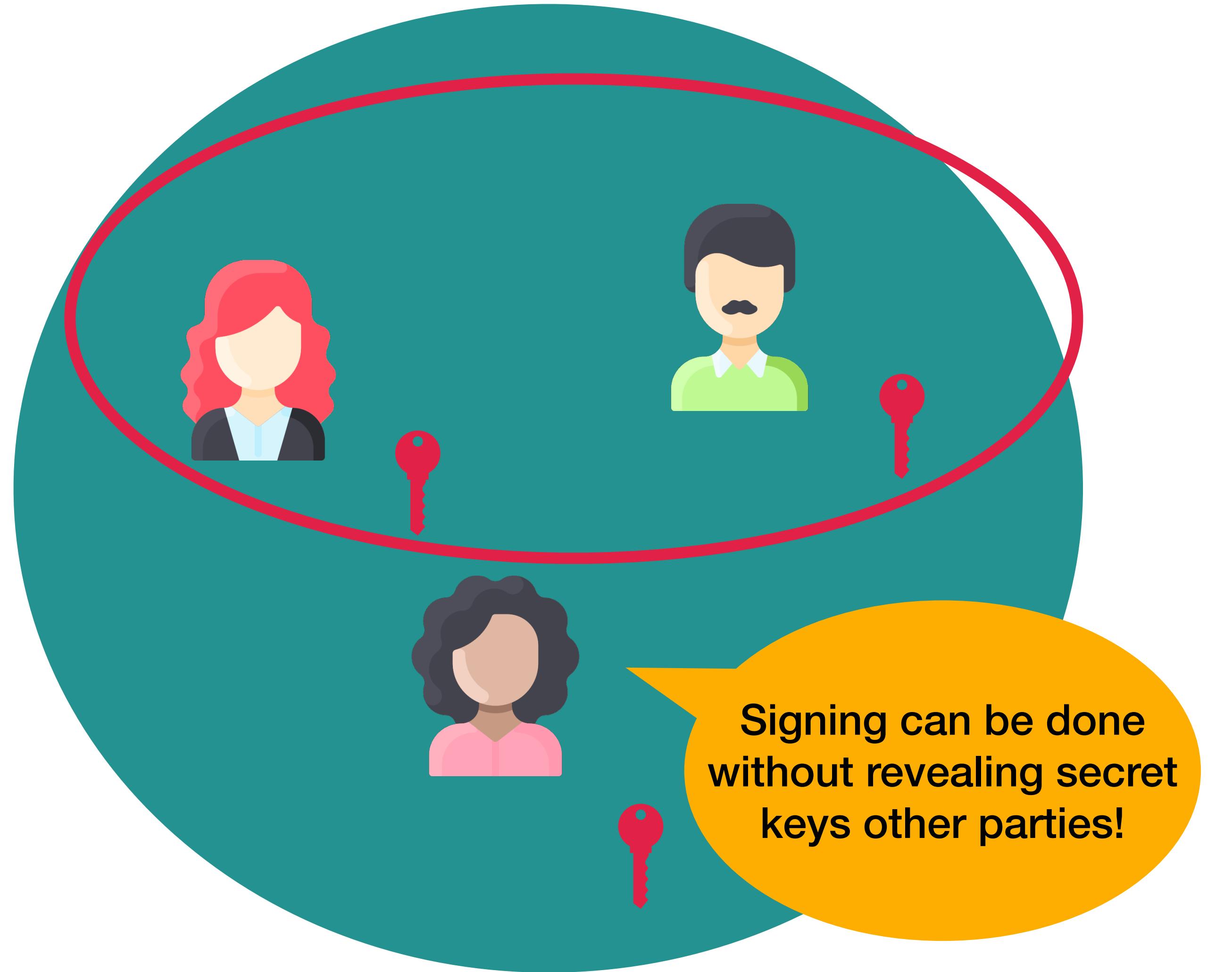


# Summary: Science is Magic



# 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]



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

# Security of Multi-Party Signatures

**Unforgeability:**

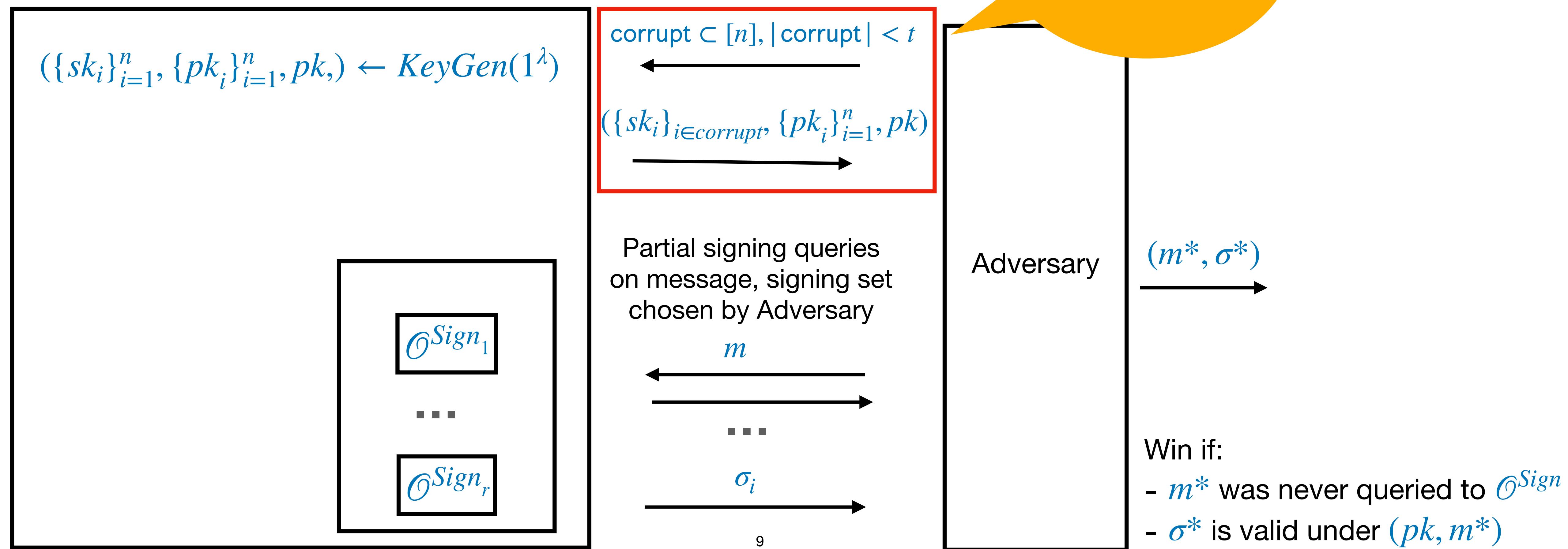
Attacker cannot forge signatures.

**Liveness:**

System can always create signatures.

# Unforgeability

A multi-party signature scheme is secure if no PPT adversary (for 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  $(PK, \sigma, m)$ :

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

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

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



zcash / zcash Public Not

Code Issues 992 Pull requests 98 Actions Projects Wiki Security

## privacy-preserving multisig transactions #782

Closed daira opened this issue on Mar 15, 2016 · 33 comments

daira commented on Mar 15, 2016 · edited

This feature would allow you to combine several independently generated proofs of ownership of distinct spending keys in order to be able to spend coins sent to a private multisig address, without requiring those keys to be brought together on the same machine.

The following desirable properties of a solution seem to be in conflict:

- no significant increase in the cost of non-multisig JoinSplits;
- indistinguishability between JoinSplits involved in multisig transactions vs non-multisig transactions.

[Edit: Pour -> JoinSplit]

1

# Existing Work

**5-Round**

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

Douglas R. Stinson<sup>1,2</sup> and Reto Strobl<sup>3</sup>

**3-Round**

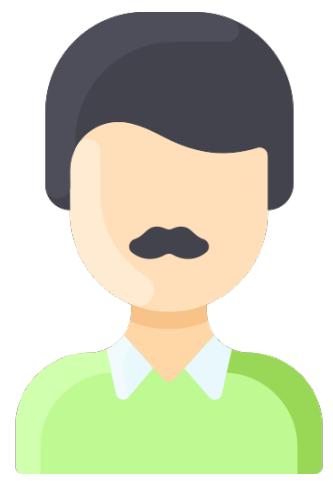
**Simple Schnorr Multi-Signatures  
with Applications to Bitcoin**

Gregory Maxwell, Andrew Poelstra<sup>1</sup>, Yannick Seurin<sup>2</sup>, and Pieter Wuille<sup>1</sup>

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

# A First Attempt

$PK_1$



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

$$R_1 = g^{r_1}$$

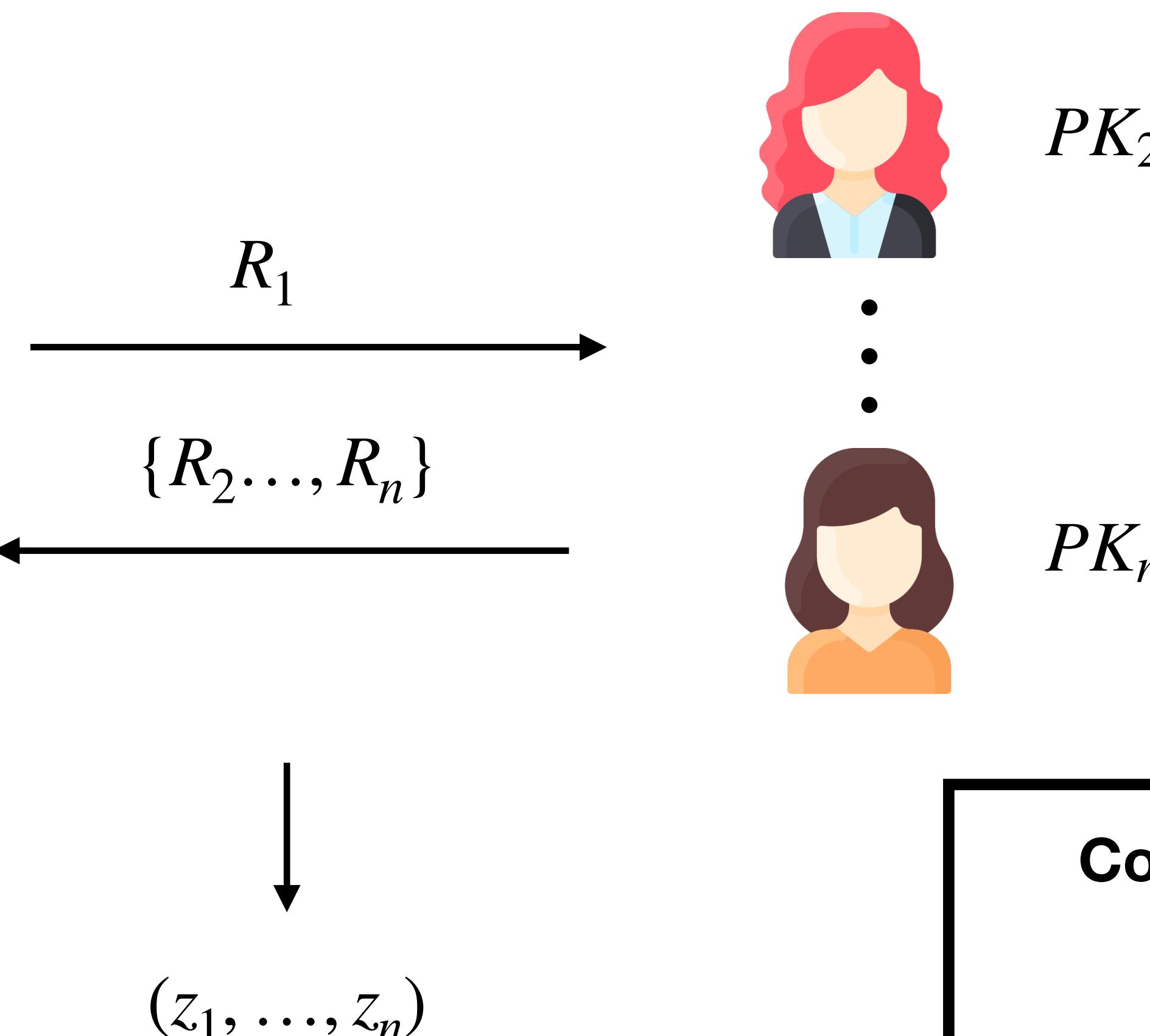
$$R = \prod_{i=1}^n R_i$$

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

$$z_1 = r_1 + csk_1\lambda_1$$

Secure in a  
sequential setting

16



$PK_2$



$PK_n$

## Combine/Verify

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

$$sig = (R, z)$$

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

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



# Concurrent Security: ROS

Session 1

$sk_1$



...

Session  $k$



$R_1^{(1)}$

$R_2^{(1)}$

$sk_2$



Can forge!

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

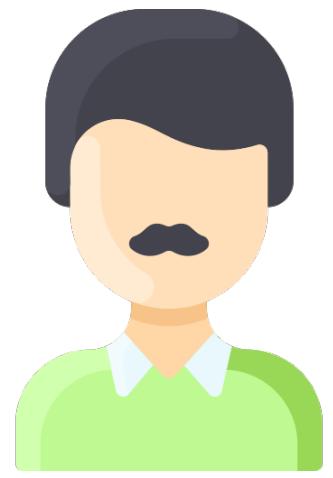
[DEFKLNS19, BLLOR21]

Affected:

- multi-signatures
- threshold signatures
- blind signatures

# One-More Forgery Adversary

$PK_1$



$$R_1 = g^{r_1}$$

$$R'_1 = g^{r'_1}$$

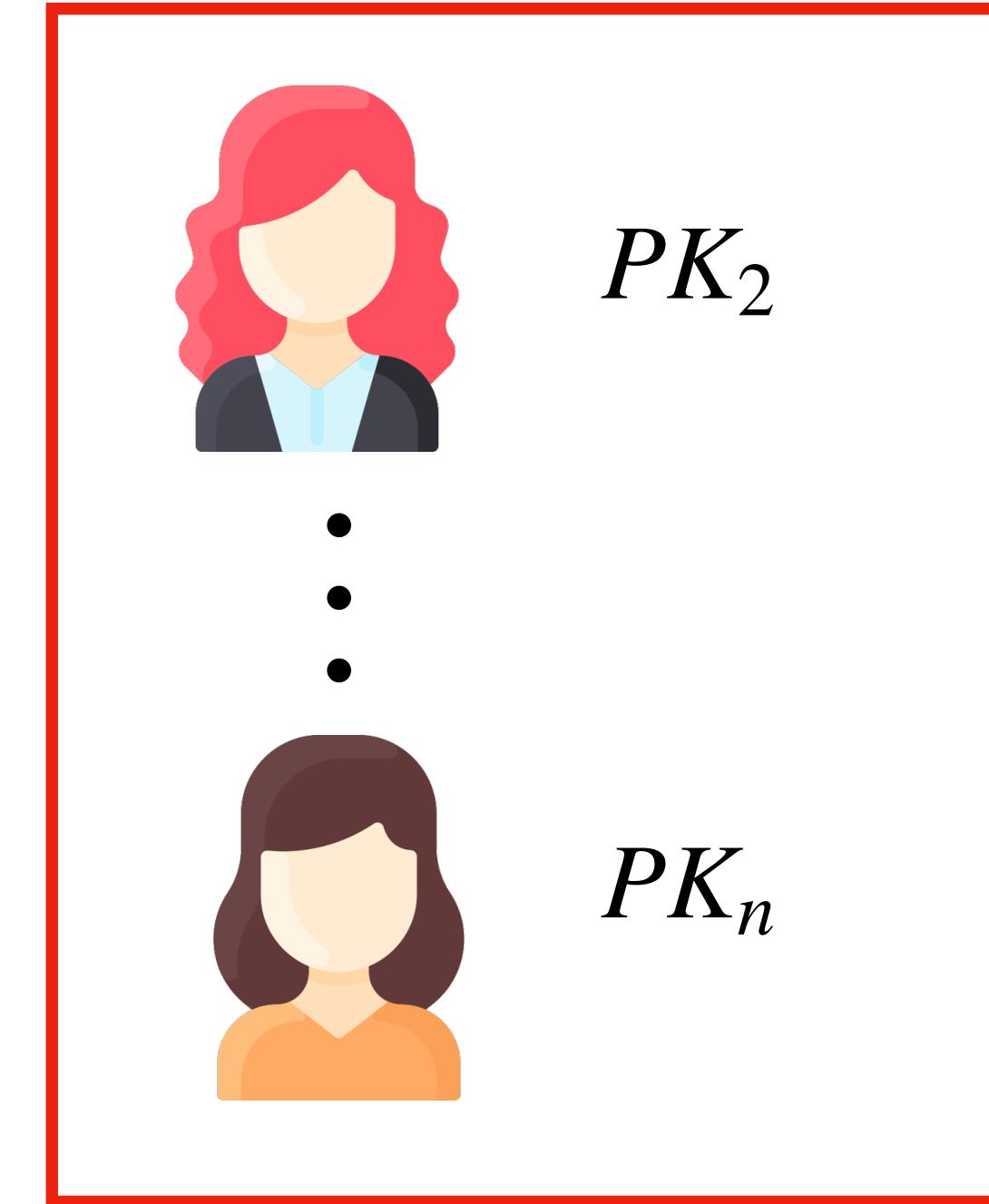
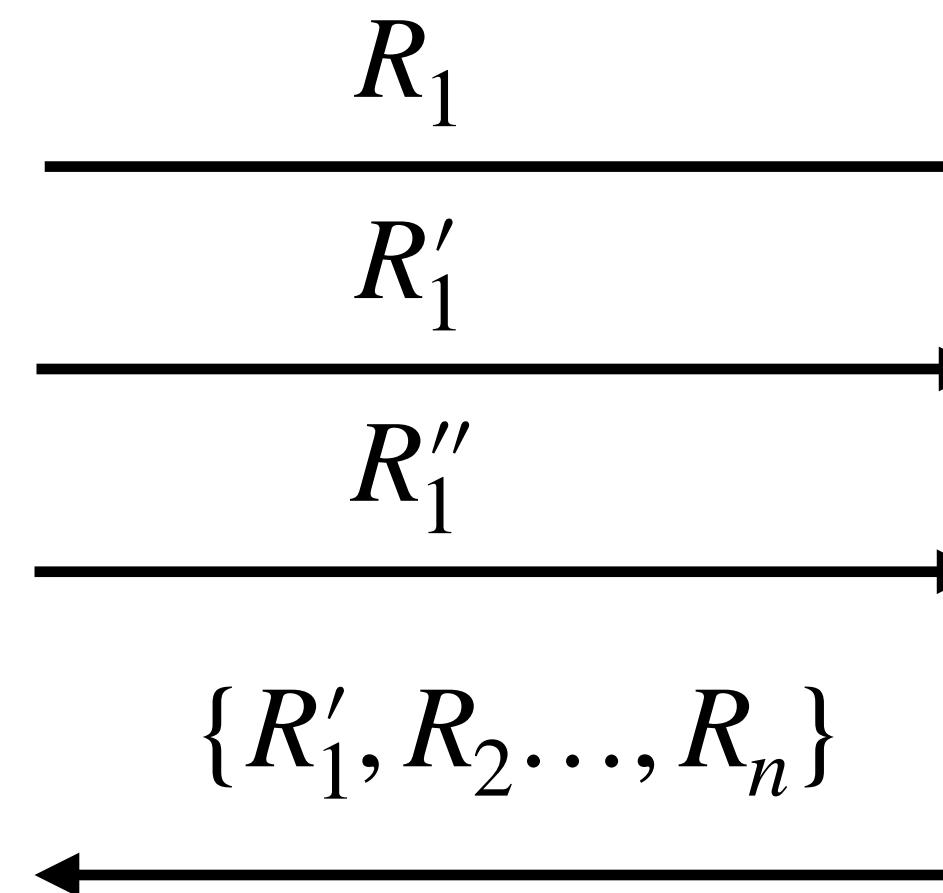
$$R''_1 = g^{r''_1}$$

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

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

$$z_1 = r'_1 + c s k_1 \lambda_1$$

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



**Challenge to Adversary:** Given  $\ell$  signing sessions, find:

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

$$z^* = \sum_{i=1}^{\ell} z_i = \sum_{i=1}^{\ell} r_i + c_i \lambda_i s k_i = \sum_{i=1}^{\ell} r_i + c^* s k$$

# Prior Insecure Schemes

Scheme	KVf	KA <sub>g</sub>	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]	$1\mathbb{G}^2$		$1\mathbb{G} + 2\mathbb{G}^2$	$1\mathbb{G}^{n+1} + 2\mathbb{G}^2$	2	$\mathbb{G}$	$\mathbb{G}^3 \times \mathbb{Z}_q^3$	$\mathbb{G}$	Insecure
MWLD [MWLD10]	$1\mathbb{G}^2$		$1\mathbb{G}^2$	$1\mathbb{G}^{n+2}$	2	$\mathbb{G}$	$\mathbb{Z}_q^3$	$\mathbb{G}^n$	Insecure
CoSi [STV <sup>+</sup> 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]	$1\mathbb{G}^2$	$1\mathbb{G}^n$	$1\mathbb{G}$	$1\mathbb{G}^2$	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 <sup>+</sup> 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 !



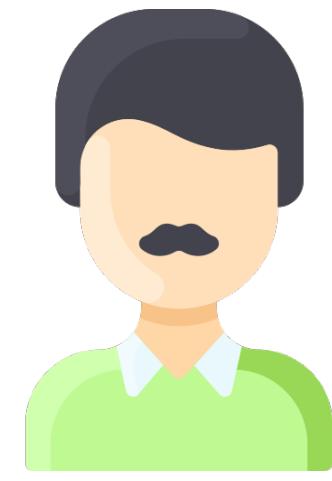
# Three Round Fix

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



# MuSig(1)

$PK_1$



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

$$R_1 = g^{r_1}$$

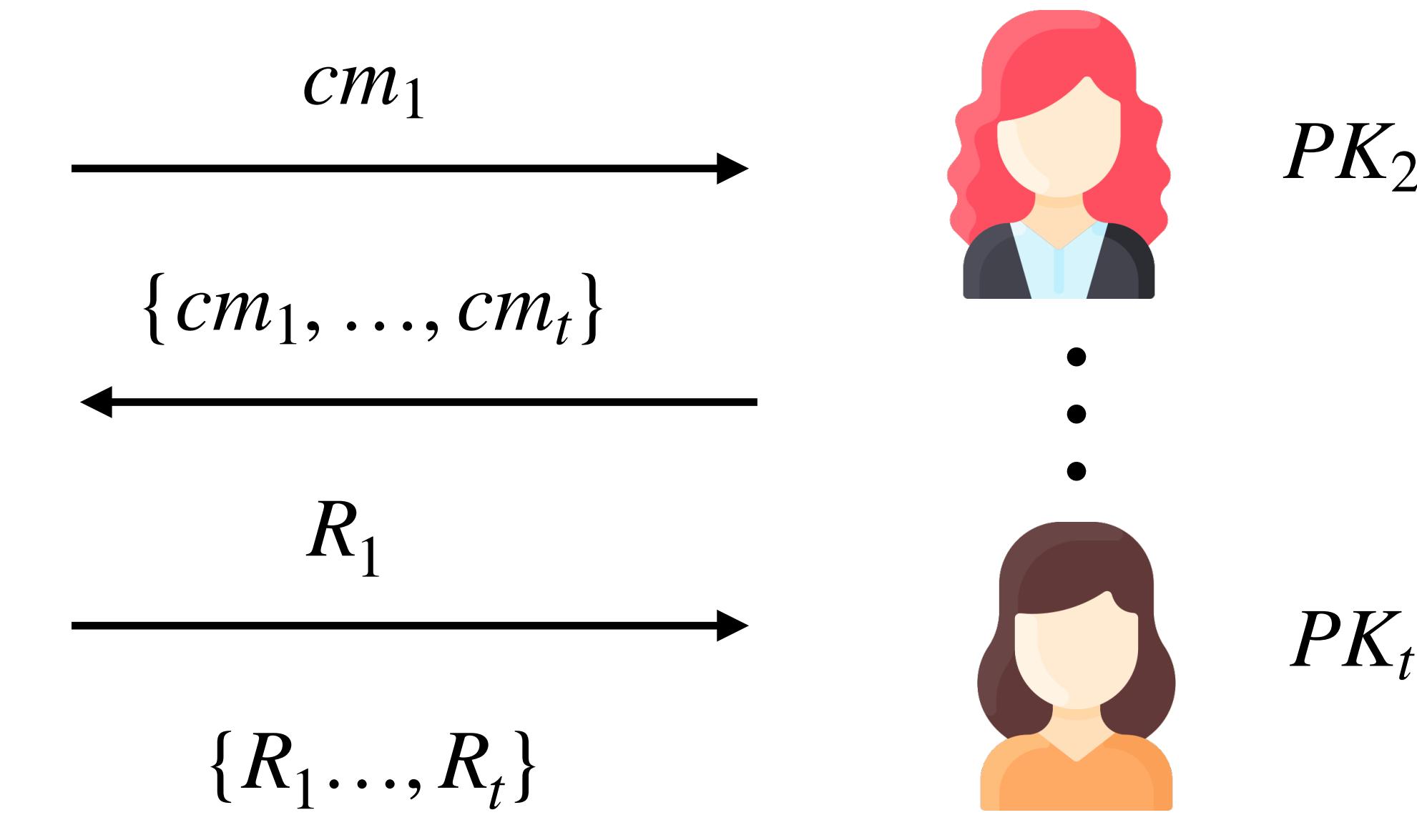
$$cm_i = H(R_i, m, S = \{1, \dots, t\})$$

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

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

$$z_1 = r_1 + csk_1$$

[MPSW18, MPSW19, CKM23, M23]



$(z_1, \dots, z_t)$

Three  
\*online\*  
rounds

**Similar Techniques:** Sparkle, ZeroS Lindell22

22

## Combine/Verify

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

$$sig = (R, z)$$

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

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



**STAND BACK**



**I'M GOING TO TRY  
SCIENCE**

**Solution #2: “Tweak” the signature.**

# FROST

# Sharing $sk$ (threshold, trusted dealer)



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

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

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

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

Shamir secret sharing

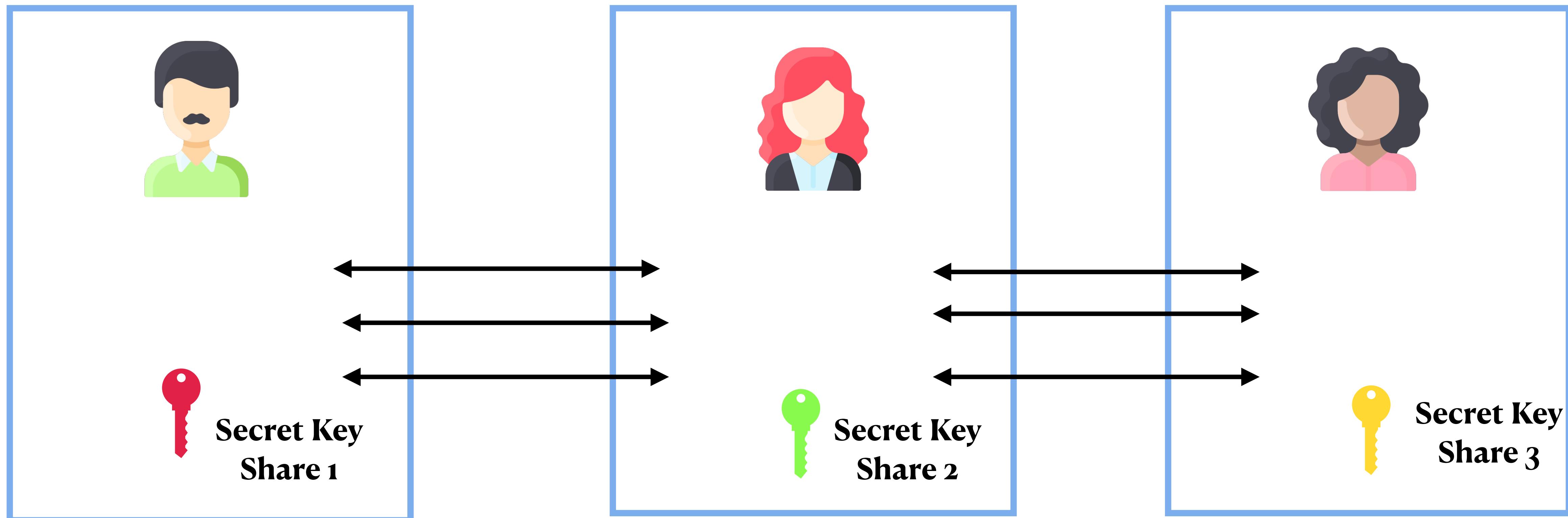
$((i, sk_i), PK)$

# Recovering $sk$

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

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

# Distributed Key Generation



No single party knows the corresponding secret key!

# FROST

[KG20, CKM21, BCKMTZ22]

- Flexible Round-Optimized Schnorr Threshold Signatures [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

# FROST

[KG20, CKM21, BCKMTZ22, CGRS23]

MuSig2  
signing is similar!  
More on this later

$PK_1$



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

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

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

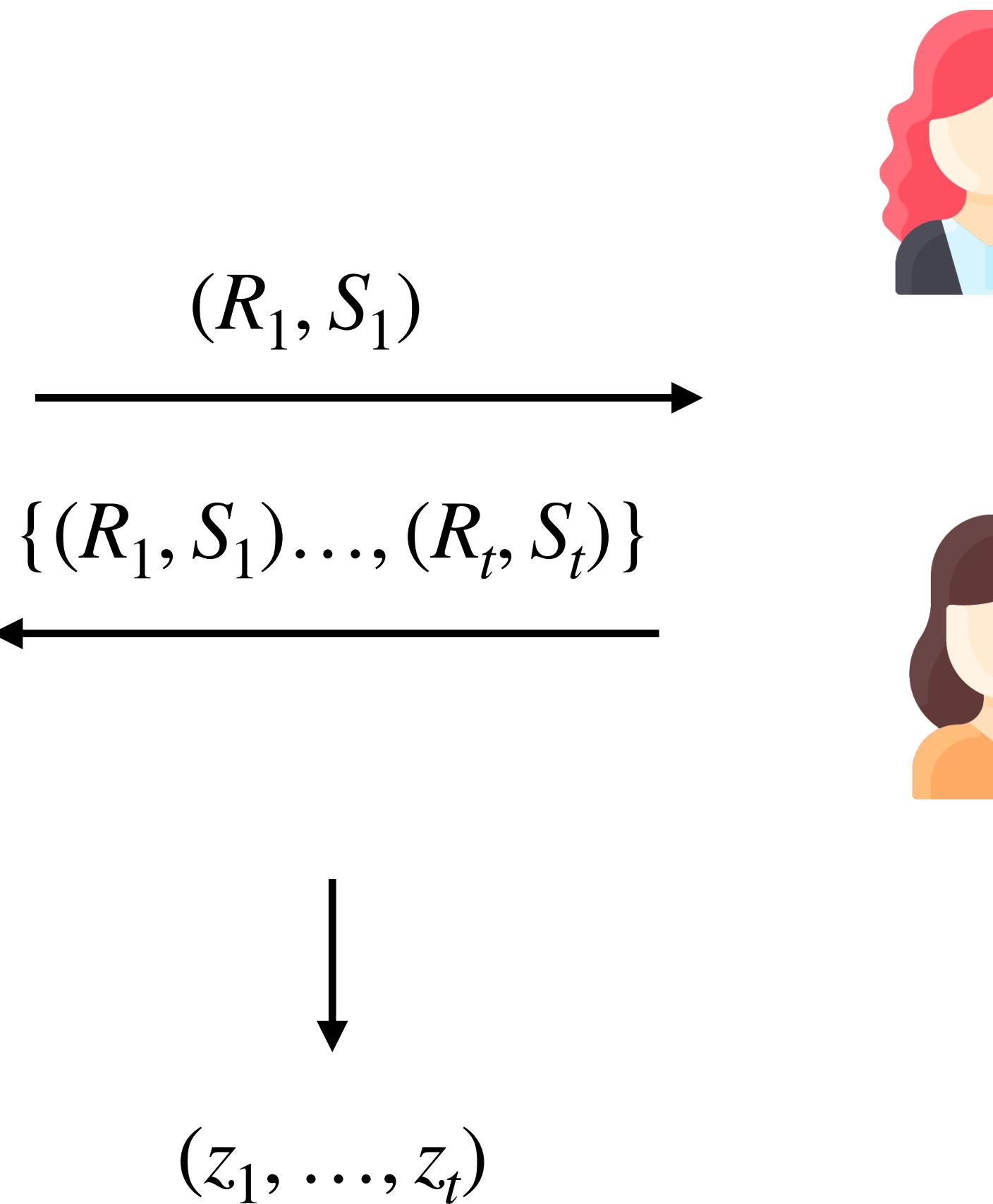
$$R = \prod_{i=1}^t R_i S_i^a$$

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

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

First Round:

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



$PK_2$



⋮



$PK_t$

## Combine/Verify

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

$$sig = (R, z)$$

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

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



# FROST: Correctness

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

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

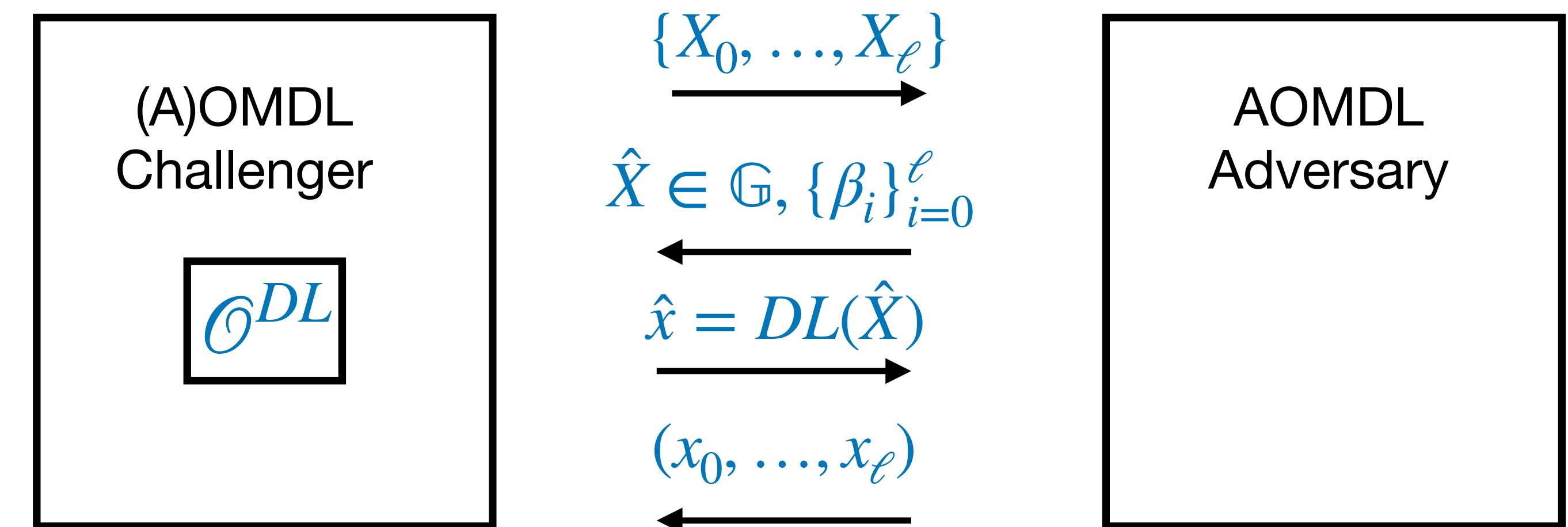
$$= \sum_{i=1}^t r_i + csk$$



By Shamir  
secret sharing

# Proving Security

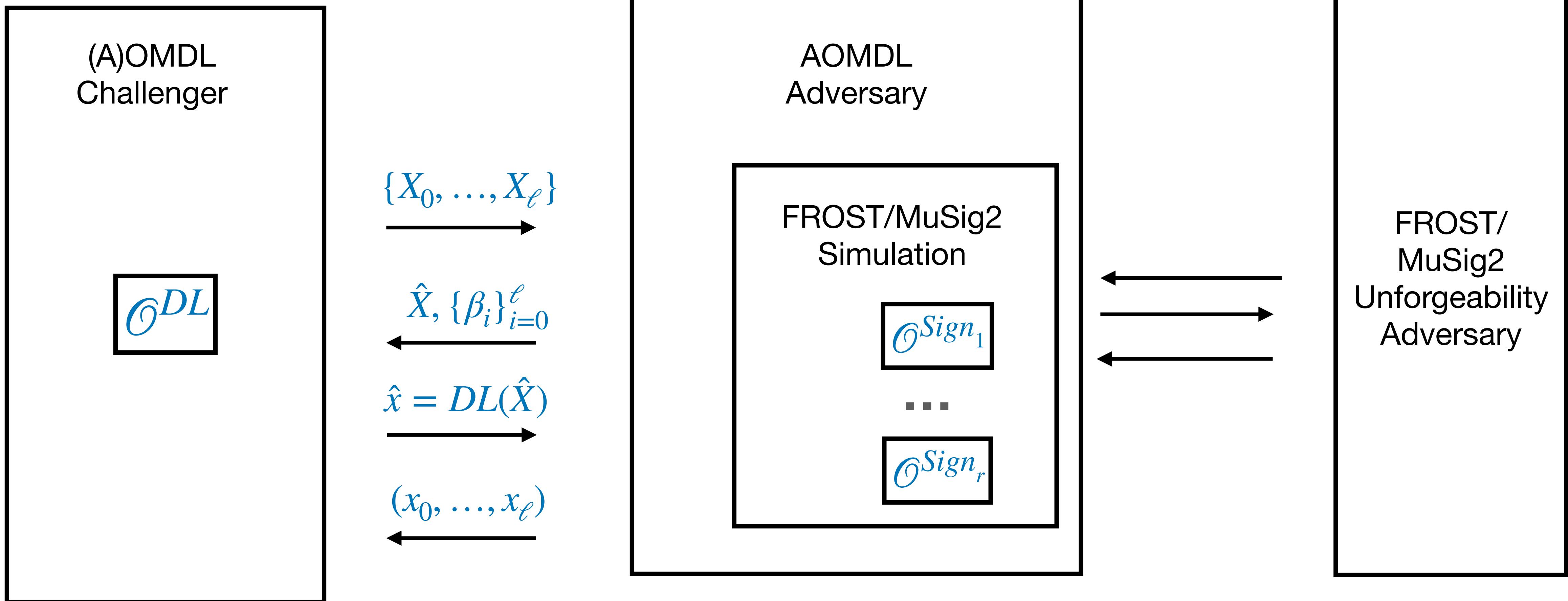
- Goal: Show that if a mathematical assumption is hard, then a scheme is secure.
- Contrapositive: If an adversary can break the security of a scheme, possible to efficiently solve a hard mathematical problem.
- AOMDL Assumption: Algebraic One-More Discrete Logarithm Assumption.
- Given  $\ell + 1$ 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



# Static Security of FROST/MuSig2

[NRS21,  
BCKMTZ22]

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



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

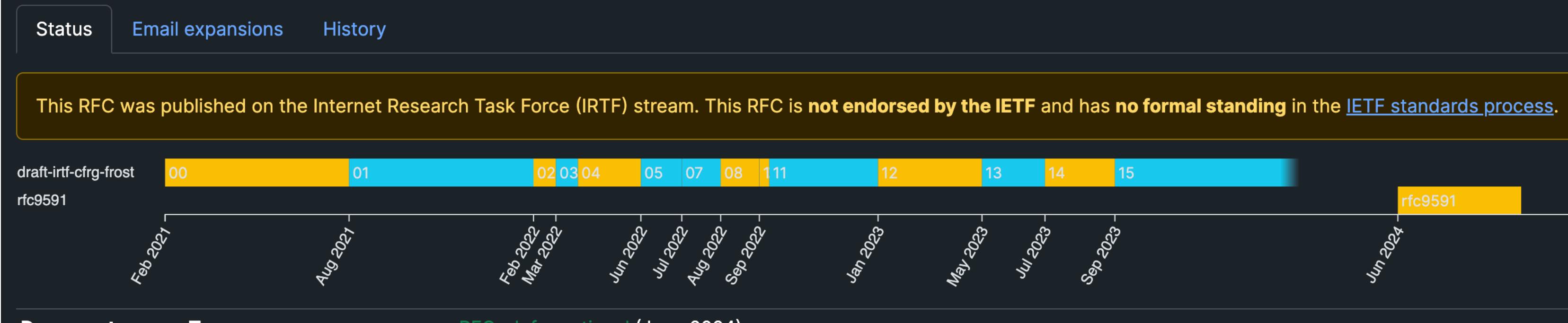
Algebraic One-More Discrete Log (AOMDL):

- stronger assumption
- + partially non-interactive schemes

# Standardization

## The Flexible Round-Optimized Schnorr Threshold (FROST) Protocol for Two-Round Schnorr Signatures

### RFC 9591



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

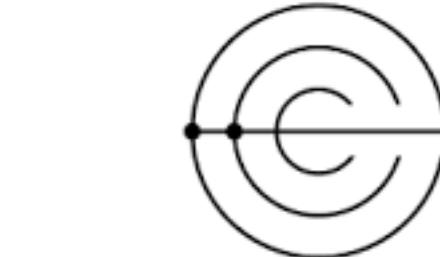
Chenzhi Zhu



web3  
foundation



ethereum  
foundation



ELECTRIC COIN CC



**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](mailto: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 Assumption Security of Multi- and Threshold FROST

Elizabeth Crites<sup>1</sup>, Chelsea Komlo<sup>2</sup>, and Mary Maller<sup>3</sup>

Robust FROST

## Better than Advertised Security for Non-interactive Threshold Signatures

Mihir Bellare<sup>1</sup> , Elizabeth Crites<sup>2</sup>, Chelsea Komlo<sup>3</sup>, Mary Maller<sup>4</sup>,  
Stefano Tessaro<sup>5</sup>, and Chenzhi Zhu<sup>5</sup> 

## A Formal Treatment of Distributed Key Generation, and New Constructions

Chelsea Komlo, Ian Goldberg, Douglas Stebila

Threshold Key  
Generation

## 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

Many faces of Schnorr

Victor Shoup  
Offchain Labs

April 8, 2024

## 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](mailto:{tessaro,zhucz20}@cs.washington.edu)

## Fully Adaptive Schnorr Threshold Signatures

Elizabeth Crites<sup>1</sup>, Chelsea Komlo<sup>2</sup>, and Mary Maller<sup>3</sup>

Threshold  
schemes with weaker  
security assumptions

## Threshold Signature from One-More Learning with Errors

Thomas Espitau<sup>1</sup>, Shuichi Katsumata<sup>1,2</sup>, Kaoru Takemure\*<sup>1,2</sup>

<sup>1</sup>PQShield

[{thomas.espitau, shuichi.katsumata, kaoru.takemure}@pqshield.com](mailto:{thomas.espitau, shuichi.katsumata, kaoru.takemure}@pqshield.com)

<sup>2</sup>AIST

## Glacius: Threshold Schnorr Signatures from DDH with Full Adaptive Security

Rena Bacho<sup>1,2</sup>, Sourav Das<sup>3</sup>, Julian Loss<sup>1</sup> and Ling Ren<sup>3</sup>

<sup>1</sup>CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

<sup>2</sup>Saarland University, Saarbrücken, Germany

<sup>3</sup>University of Illinois at Urbana Champaign

[{renas.bacho, loss}@cispa.de, {souravd2, renling}@illinois.edu](mailto:{renas.bacho, loss}@cispa.de, {souravd2, renling}@illinois.edu)

## Practical Schnorr Threshold Signatures Without the Algebraic Group Model

Hien Chu<sup>1</sup>, Paul Gerhart<sup>1</sup>, Tim Ruffing<sup>1</sup>

<sup>1</sup> Friedrich-Alexander-Universität Erlangen-Nürnberg

<sup>2</sup> Blockstack

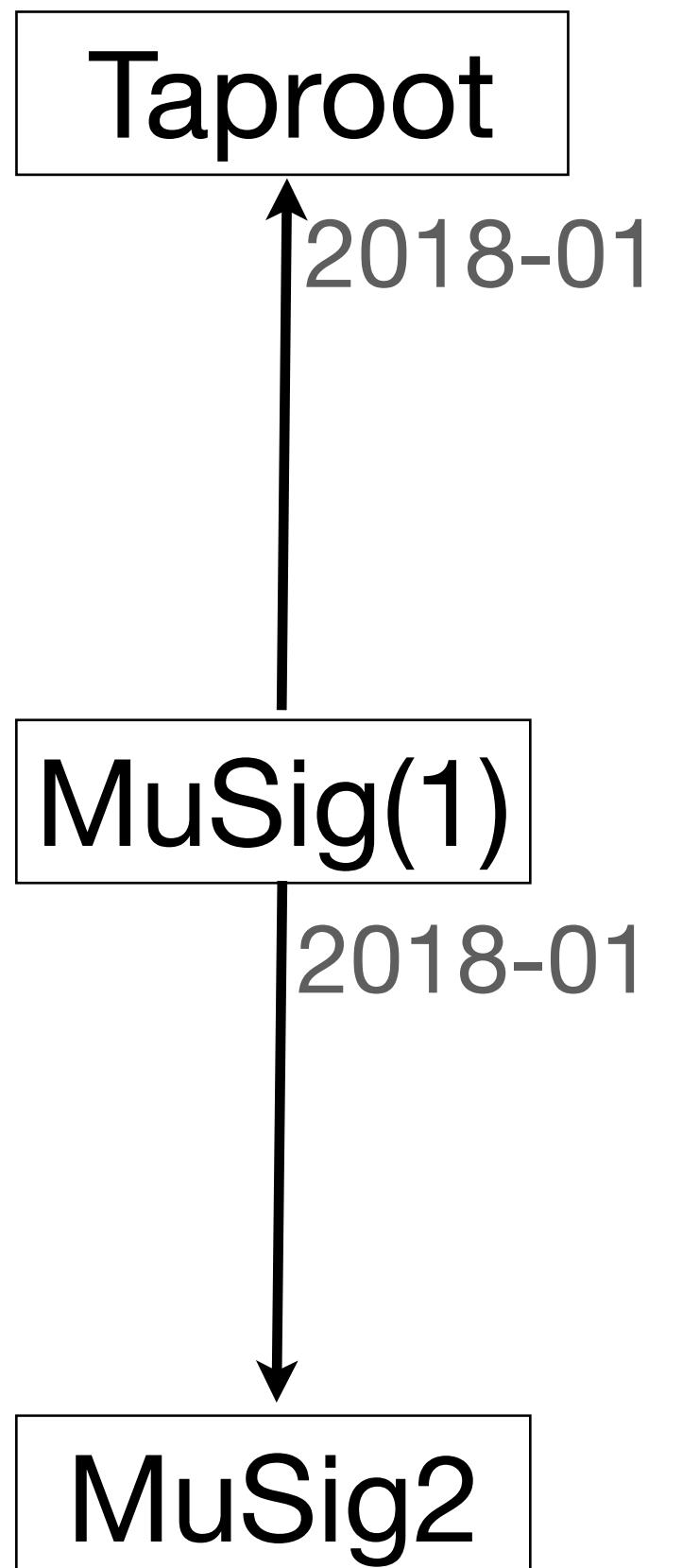
Post Quantum (Lattice)  
MuSig2 and FROST

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

Cecilia Boschini<sup>1</sup> , Akira Takahashi<sup>2</sup> , and Mehdi Tibouchi<sup>3</sup> 

# Towards MuSig2

# MuSig & Bitcoin



2021-07

Eurocrypt 2024

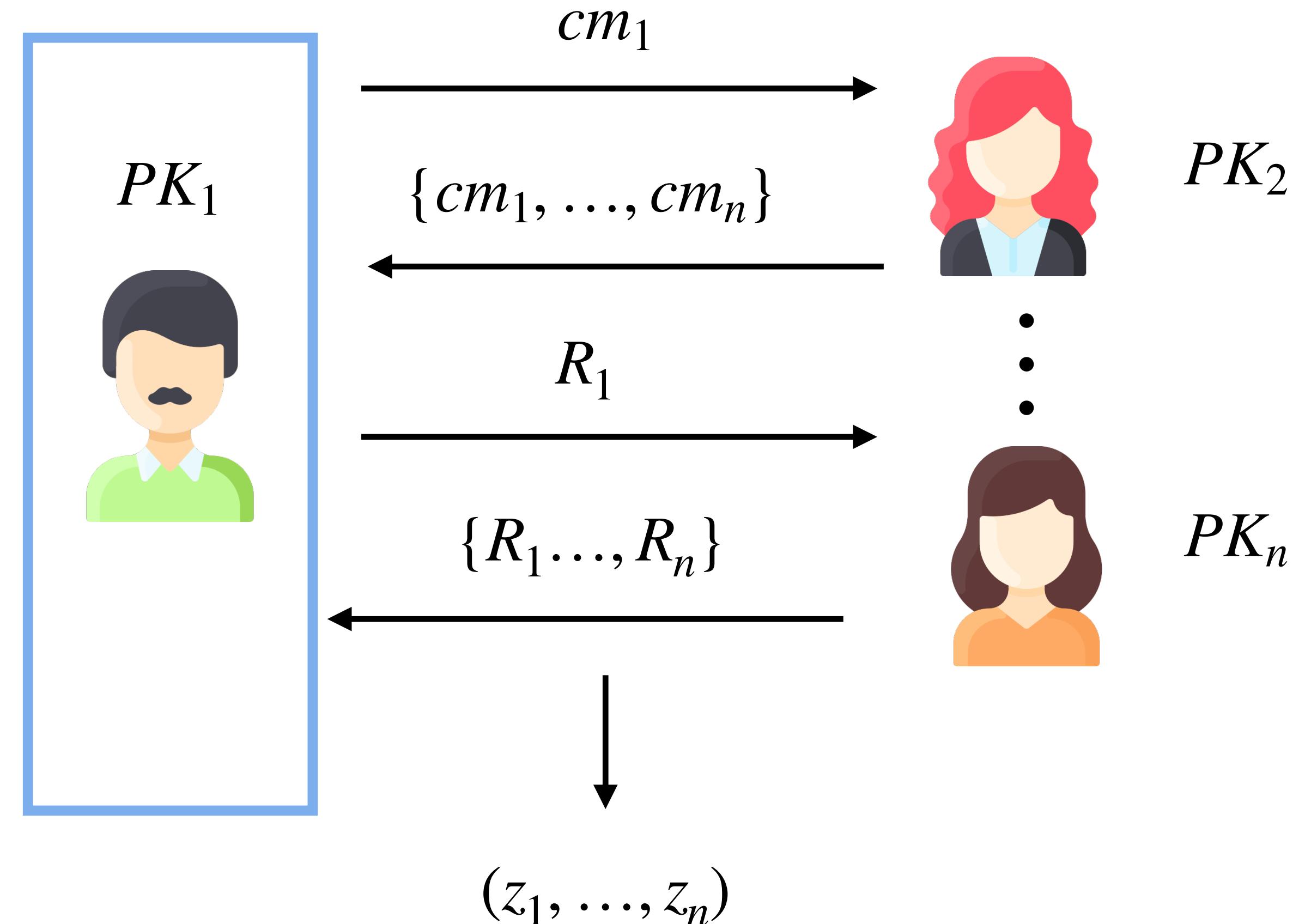
# MuSig(1) [MPSW19]

## Simple Schnorr Multi-Signatures with Applications to Bitcoin

- First **Schnorr signature-compatible** multi-signature scheme with non-interactive, **public key aggregation**.

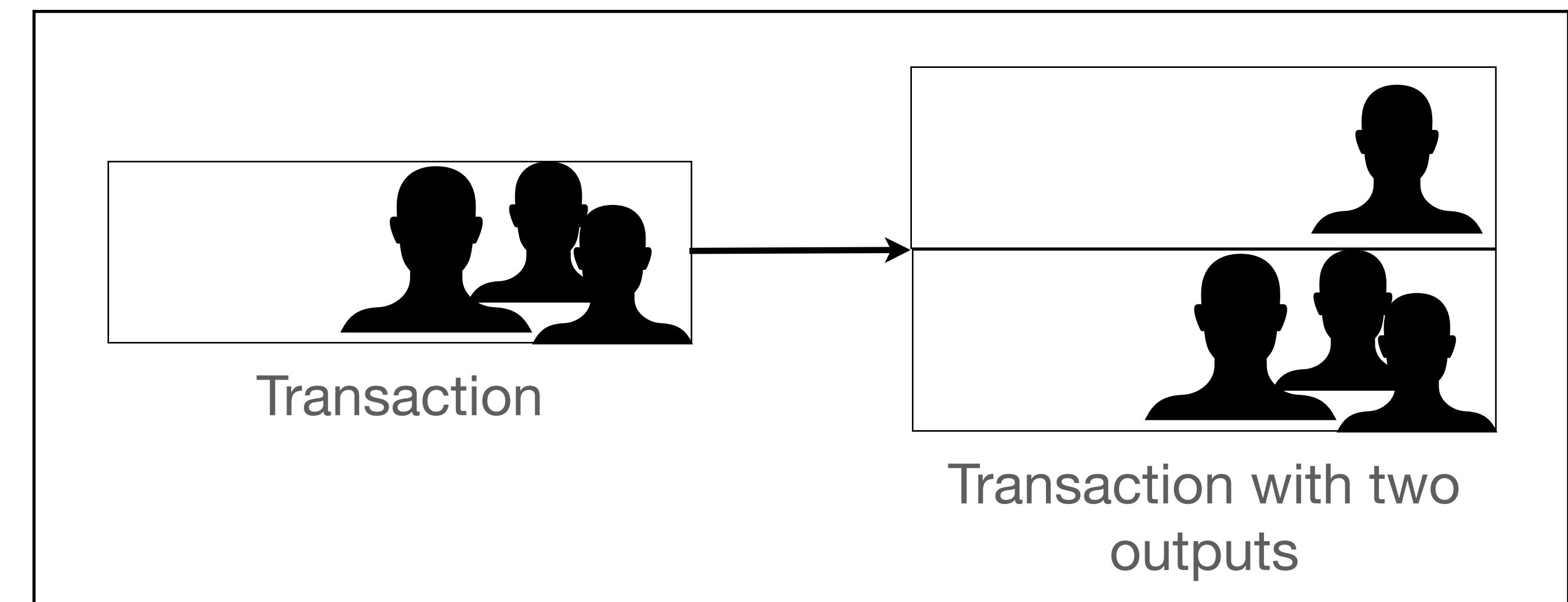
- $\widetilde{PK} = \text{KeyAgg}(\{PK_1, \dots, PK_n\})$

- "[Bitcoin] moving to Schnorr signatures would definitely help deploying compact multi-signatures"



# MuSig(1) Motivation for Bitcoin

- Bitcoin has always supported "trivial multi-signatures"
  - $PK = (PK_1, \dots, PK_n)$ ,  
 $sig = (sig_1, \dots, sig_n)$
- 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

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

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

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

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

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  $\text{taptweak}$ .
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

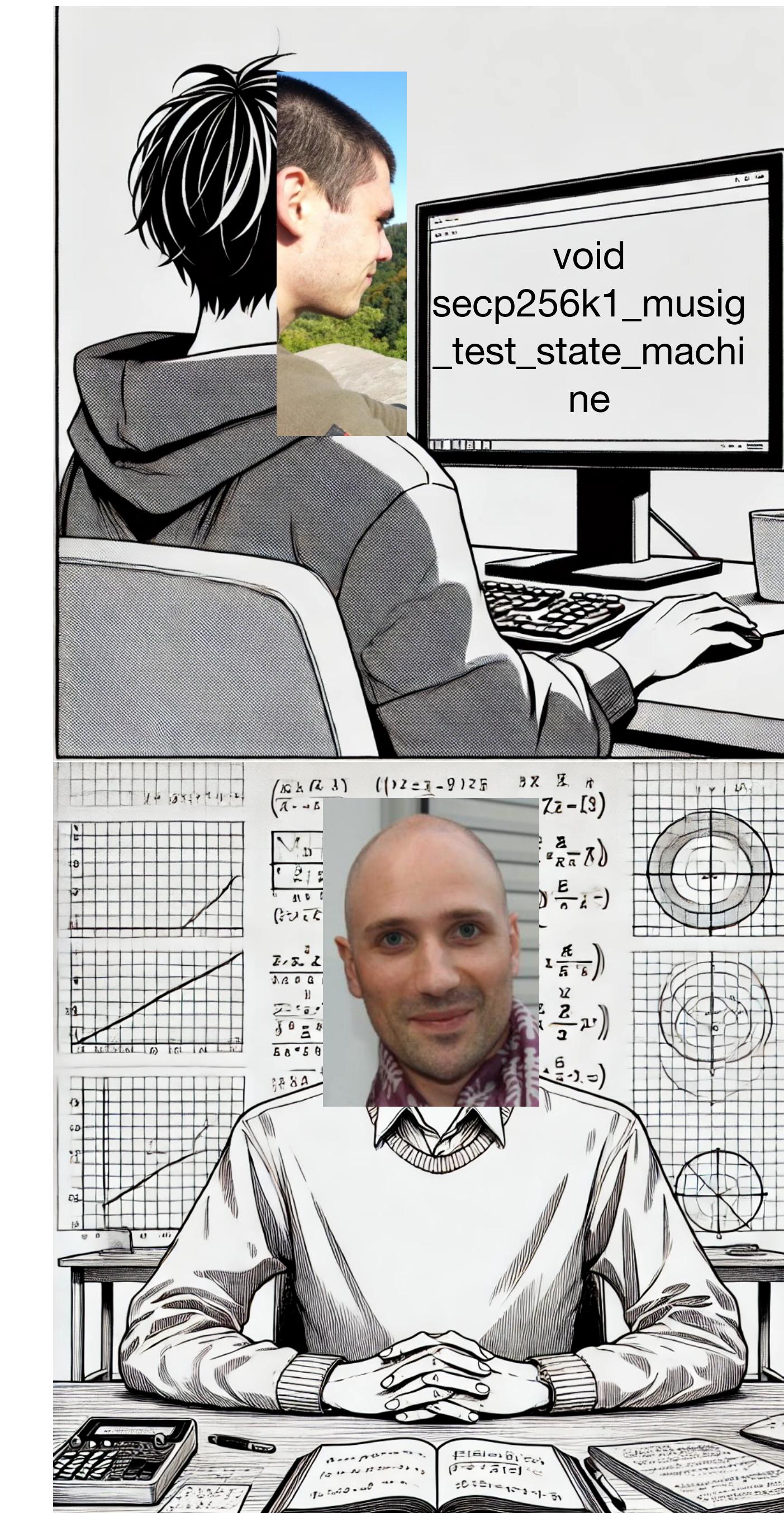
"[...] it is almost always the case that interesting scripts have a logical top level branch which allows satisfaction of the contract with nothing other than a **signature by all parties**."

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.

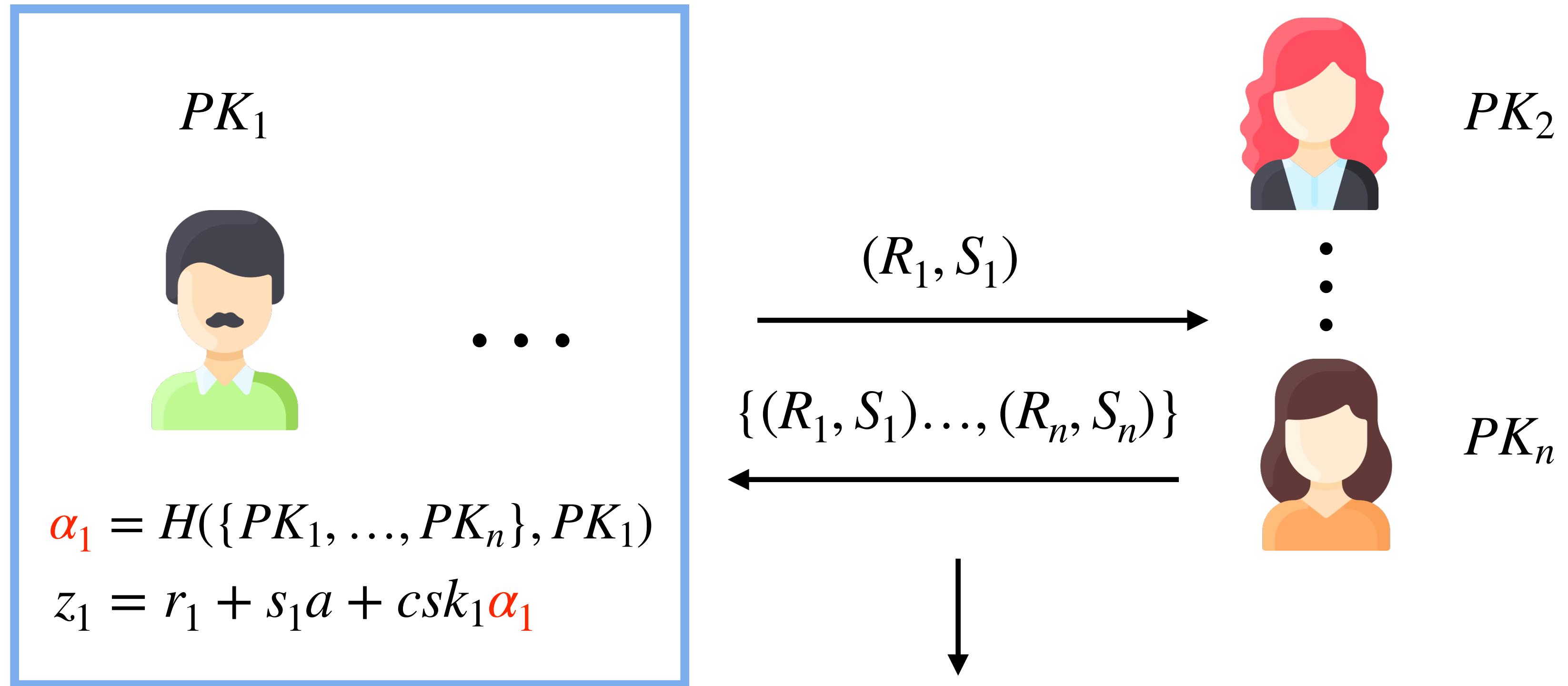
# MuSig2 Origin



# MuSig2 [NRS21]

## Simple Two-Round Schnorr Multi-Signatures

$$\begin{aligned}\widetilde{PK} &= \text{KeyAgg}(\{PK_1, \dots, PK_n\}) \\ &= \prod_{i=1}^n PK_i^{\alpha_i}\end{aligned}$$

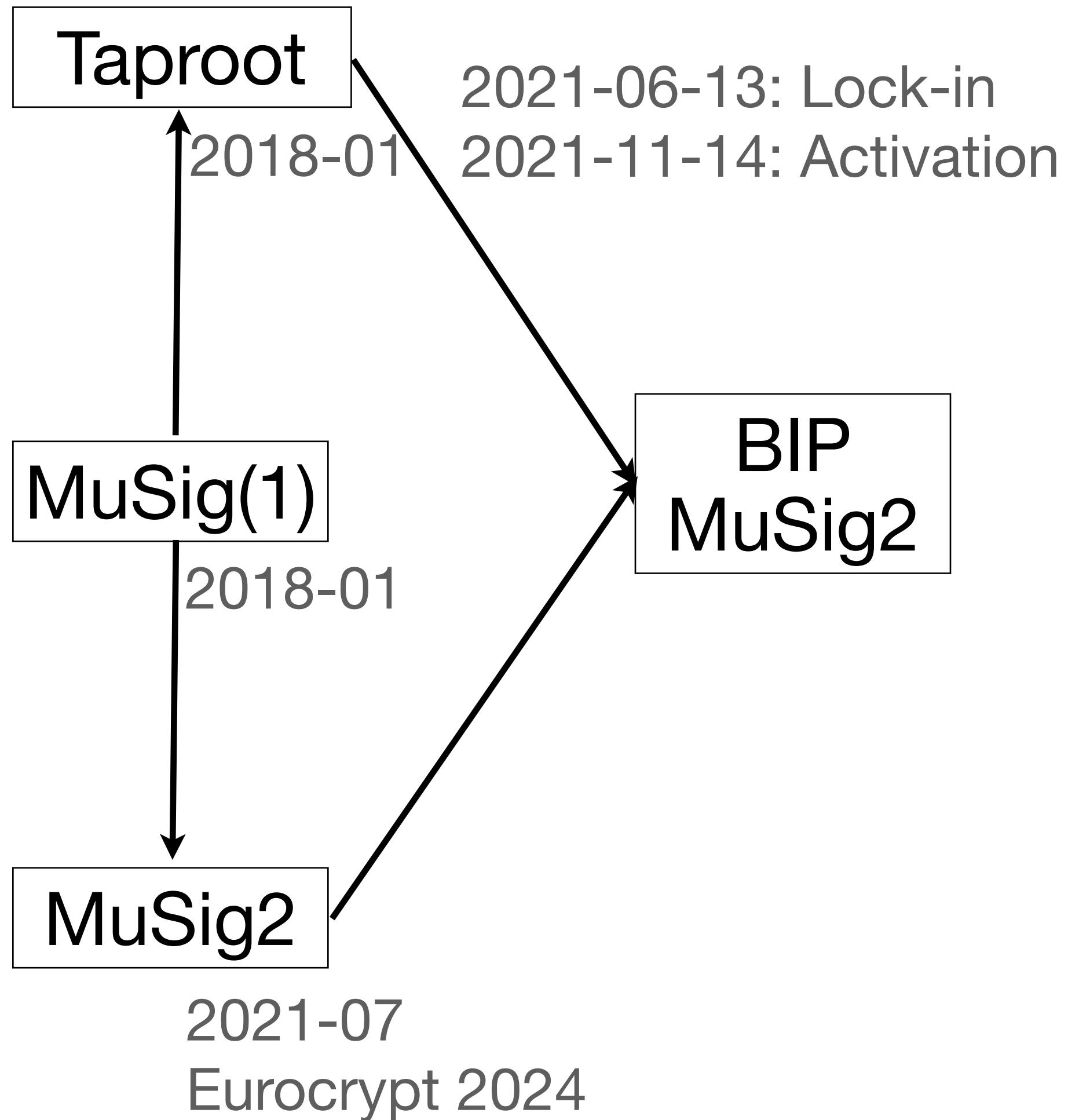


First Round:

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

# MuSig2 in Practice

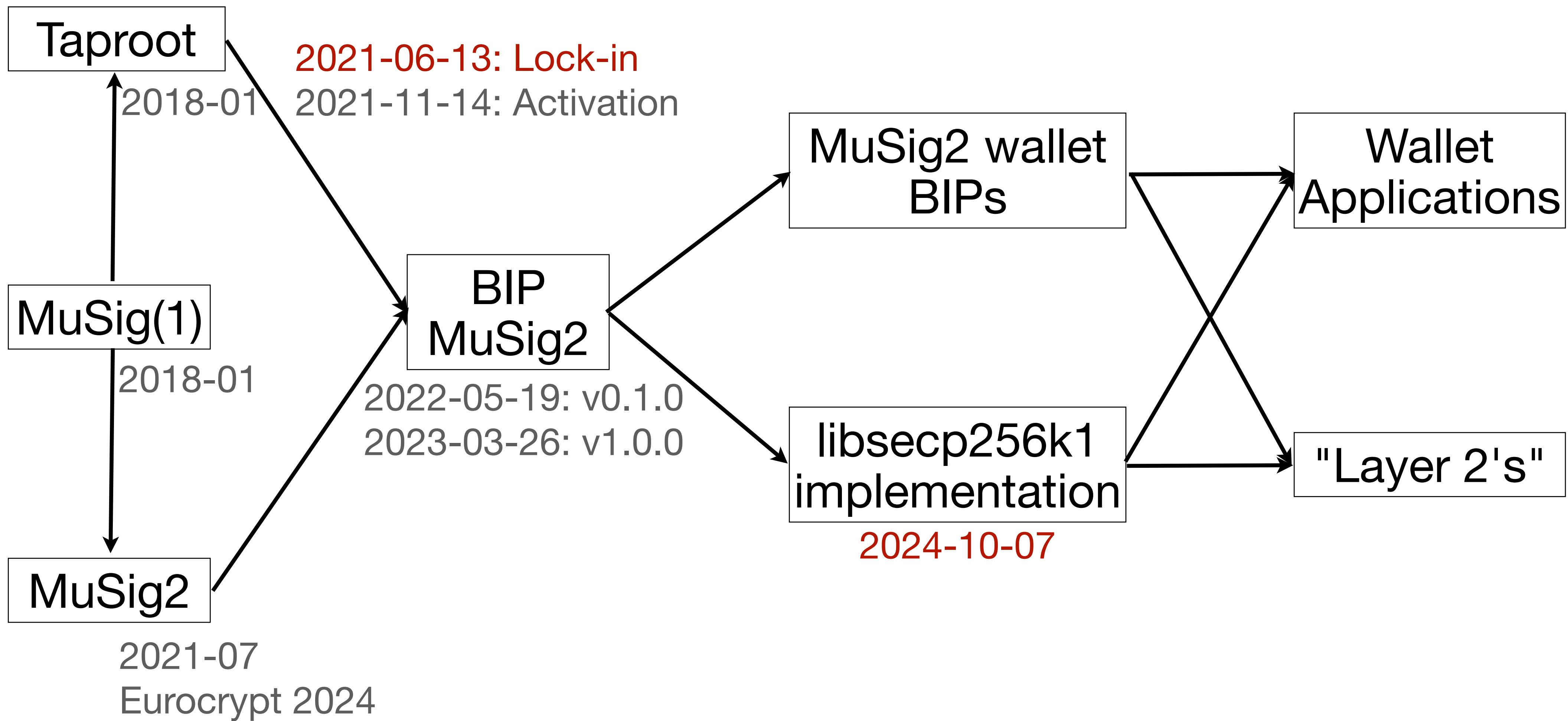
# MuSig & Bitcoin



# 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 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:

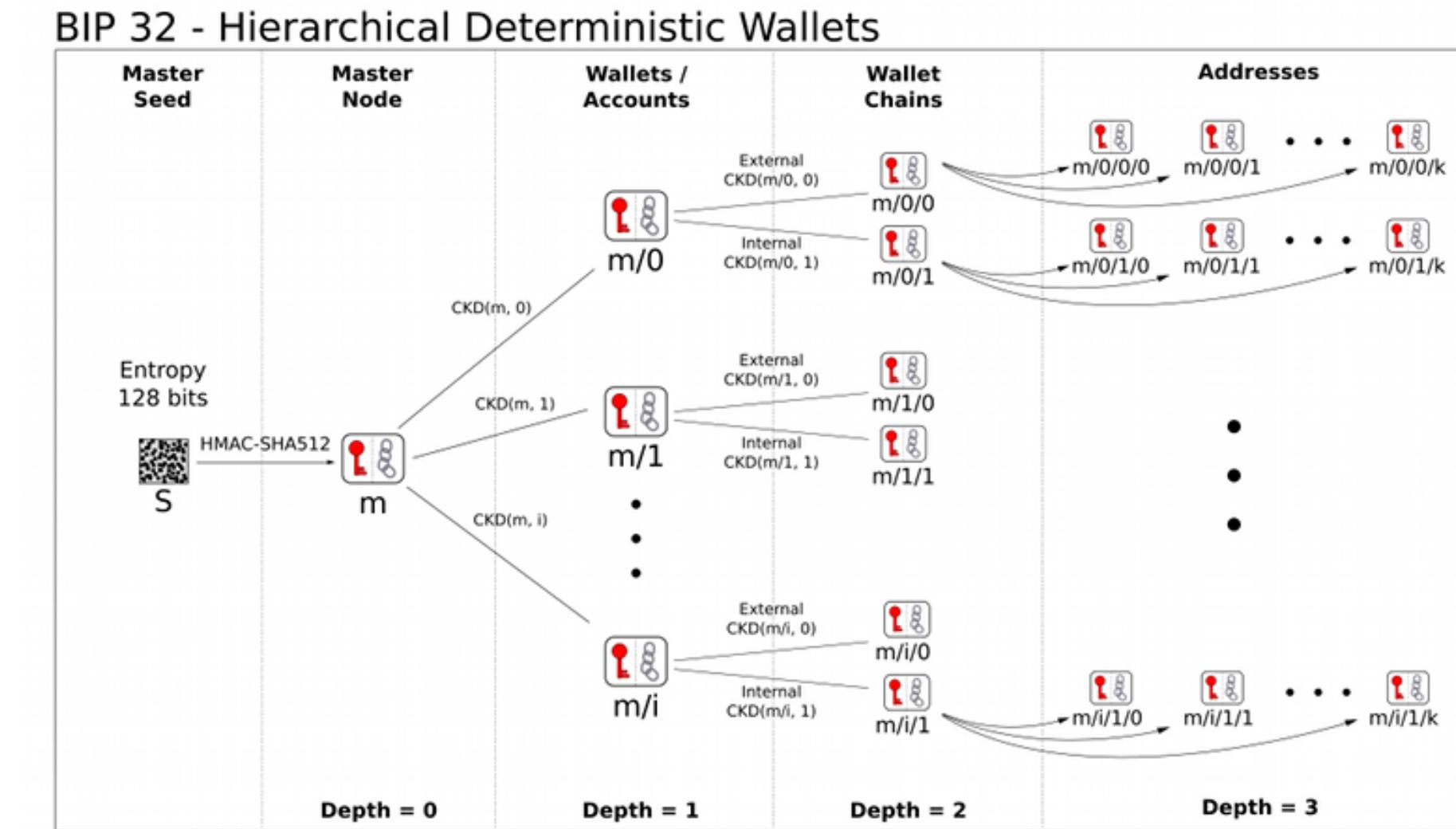
- Need to get **feedback from experts**; experts tend to be busy with other things
- Need to get **feedback from potential users**
  - **Implementations take time** and usually they don't happen until BIP/library is finalized
  - Limited urgency due to **low network fee pressure**
- Maximize **clarity** 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 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

commented on Jan 16

Contributor ...

Not actively reviewing, but trying to use this.

The `new_musig_nonce_pair` function has this sentence in the documentation that is not correct english and confuses me:

If you cannot provide a `sec_key`, `session_id` UNIFORMLY RANDOM AND KEPT SECRET (even from other signers).

This makes me wonder, the name "SessionId" somehow made me think it had to be shared between everyone in the same signing session. But from that sentence maybe it seems that it should be locally random and not shared?

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_senonce {
```

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

2. The `secp256k1_musig_senonce` structure is never copied or serialized.

See also the comment on `secp256k1_musig_senonce` in `include/secp256k1_musig.h`.

# Can we accelerate the development timeline?

## Misuse-resistant API Design is Important

Jonas @jonas 6:53 PM

Hey [REDACTED] you mentioned you're planning to store the nonces in the wallet. What happens if someone restores an old wallet (or wildly copies the wallet file around)? Wouldn't that mean that the user may reuse their nonce?

- 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

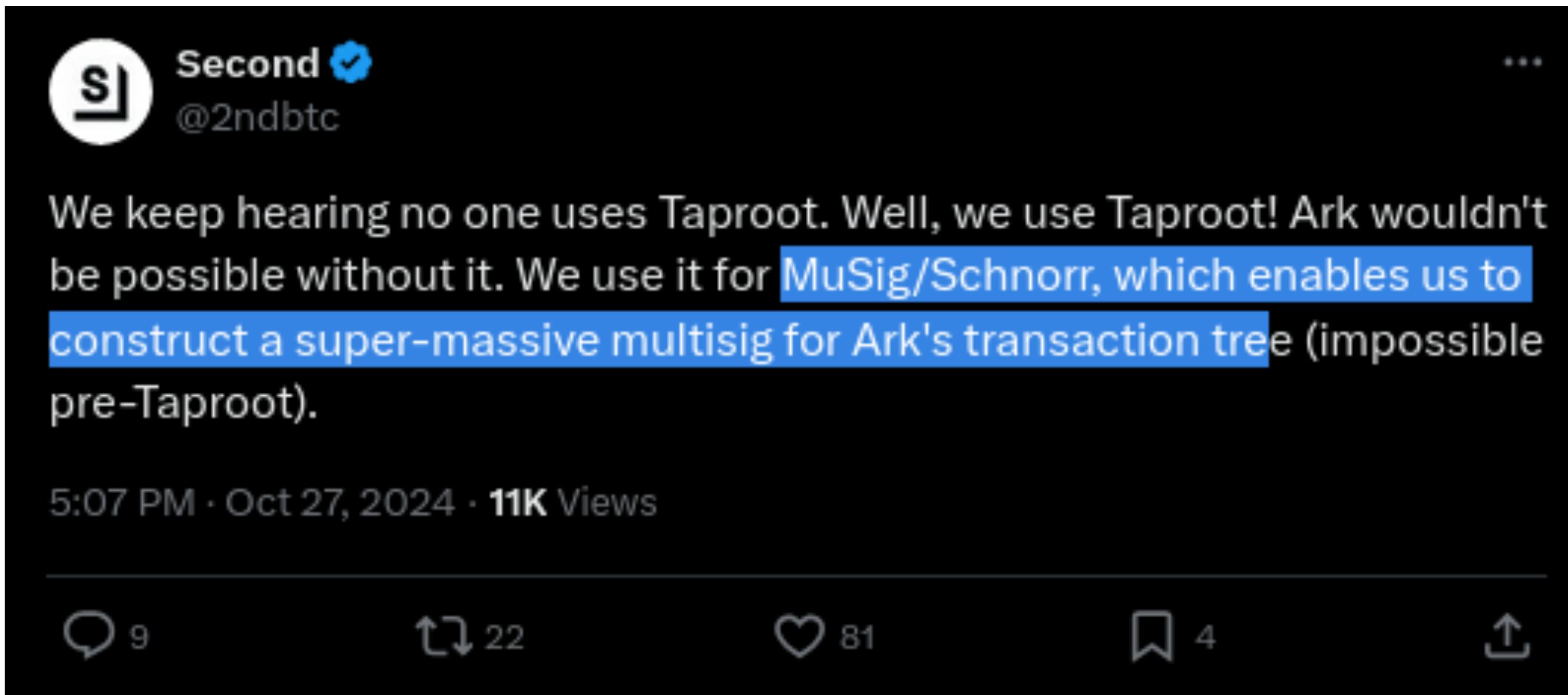
# Applications

	Application	
<b>In Production*</b>	Wallet with third-party programmable cosigner	2-of-3 (using Taproot tree)
	Swap between Lightning Network and on-chain.	2-of-2
<b>In Development</b>	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

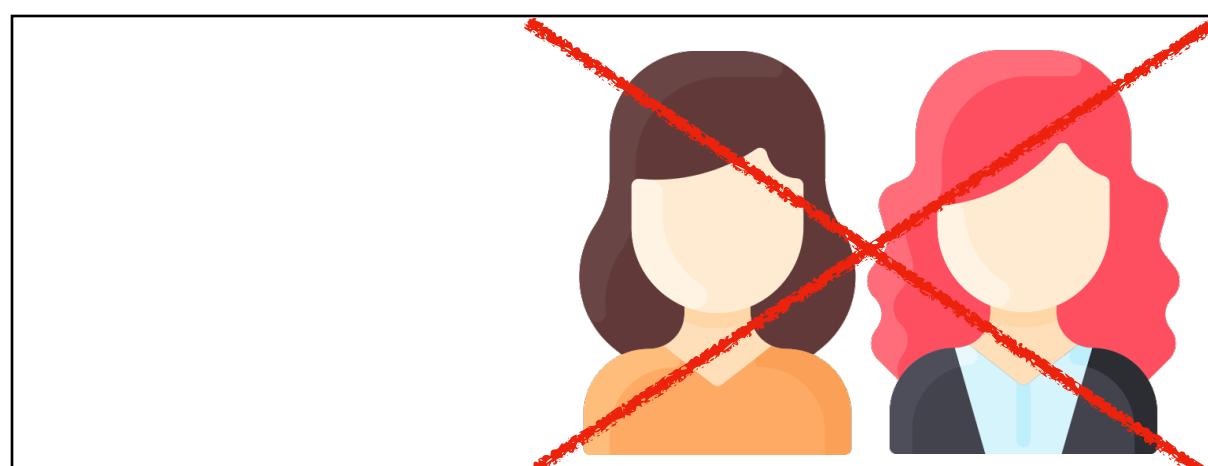
Model	Number of Nonces
ROM	4
ROM + AGM	2

- 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.

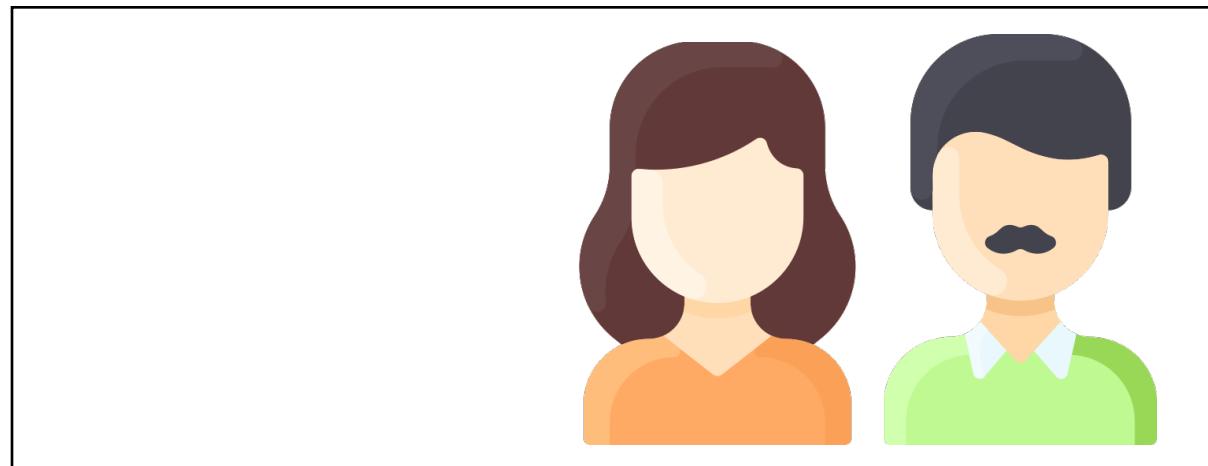
# More Applications

## 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