Çok Taraflı Eşik İmzalar
Multiparty Threshold Signatures
Promises and Challenges

Muhammed Ali Bingol

muhammed.bingol@dmu.ac.uk

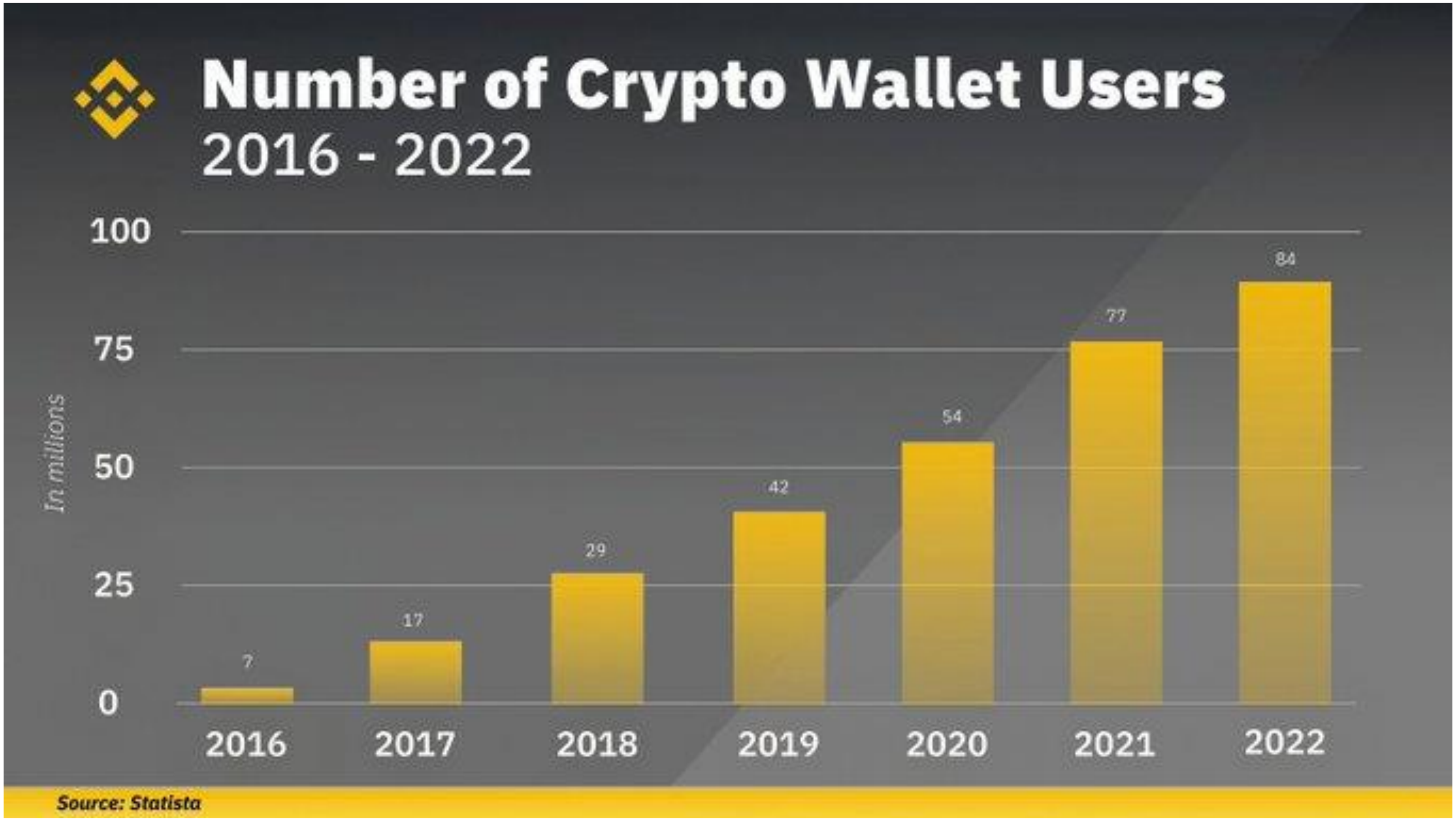
dmu.ac.uk
DE MONTFORT UNIVERSITY
LEICESTER

May 7, 2023

Outline

- Motivation
  - Self Custodian Systems
  - April 2023 Wallet attacks
  - Hot-Warm-Cold Wallets
- Secret Sharing
  - m-out-of-n Threshold Secret Sharing
  - Threshold signatures
- Signature Lengths
- BLS Signature Based TSS
  - Signature aggregation
  - MultiSign
  - Threshold BLS
- Schnorr Signature based TSS
- (EC)DSA Based TSS
  - MtA – Multiplicative to additive
- FROST

Self Custodian Systems



April 18th , 2023 –Metamask Attack

METAMASK

Featu

I don't know how big it is but since Dec 2022 it's drained 5000+ ETH and ??? in tokens / NFTs / coins across 11+ chains.

Unfortunately, transactions cannot be reversed, nor missing funds restored. MetaMask is a self-custodial wallet, which means we cannot control access to user accounts, nor intervene and rescue your account or funds for you.

Why did this happen?

Due to the sheer scale and scope of web3, there is an enormous abundance of attack vectors that could have been the reason your wallet was compromised. Some common causes are listed below:

- Your computer has been compromised with malicious software and you stored your private information on your computer, allowing it to identify and retrieve your Secret Recovery Phrase, for example.
- You have visited a malicious phishing website that stole your information.
- You gave your private key or Secret Recovery Phrase to someone or a site.
- You gave a dapp or site's smart contract unlimited access to your funds (find out how to revoke access here).
- You installed a fake MetaMask extension that stole your funds.

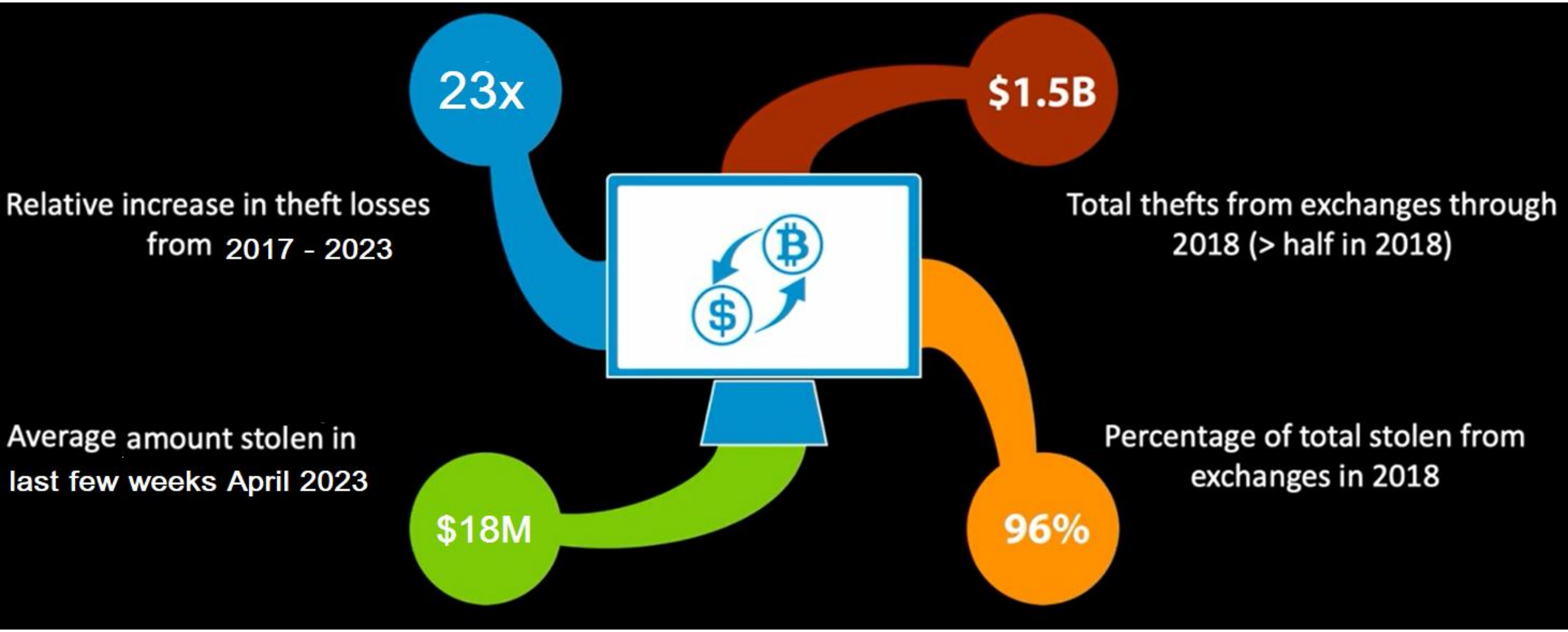
To learn about types of scams that you may have encountered, check out our Staying Safe in Web3 section.

More attacks on wallets April 25th, 2023

25 APRIL 2023 / DONJON

Funds of every wallet created with the Trust Wallet browser extension could have been stolen without any user interaction

Crypto Thefts: Significant and Increasing



Sepior©

Hot-Warm-Cold Wallets

Crypto Currency Security

Crypto Currency Security = Private Key Security

How to share a secret?

<u>Informally</u>:

-Avoid single point of failure

We want to share a secret x between a group of parties, in such a way that:

1. any set of up to **t** corrupted parties has no information on x, and
2. even if **t** parties refuse to cooperate we can still reconstruct the secret x.
3. Hierarchical Threshold Secret Sharing is also possible

m-out-of-n threshold secret sharing

we will have
m = t+1

dealer's secret **x**

(n = 5)

x_1 **x_2** **x_3** **x_4** **x_5**

1. Every set of at least **m** players (t+1 or more) can **reconstruct** x.
2. Any set of less than **m** players (t or less) has **no information about** x.

Shamir's secret sharing [1/2]

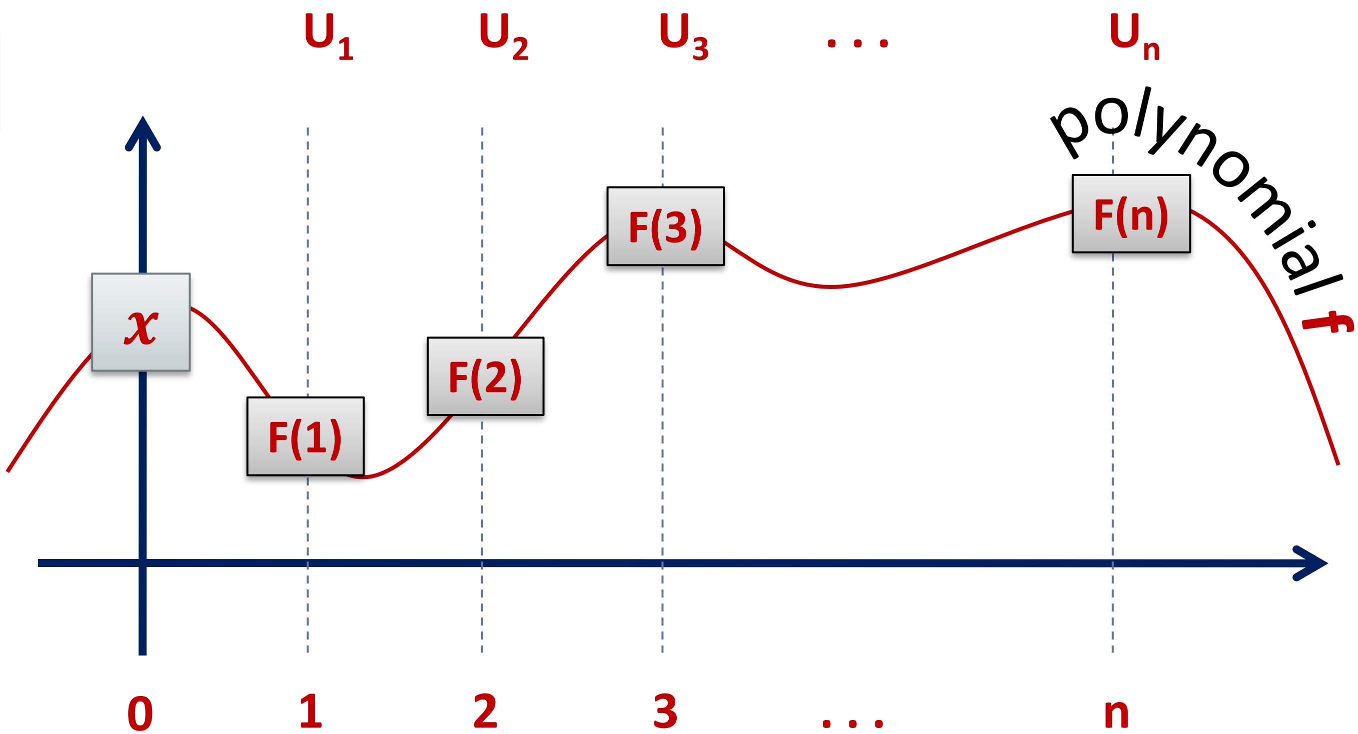
Pick random *coefficients* $a_1, a_2, \dots, a_{k-1} \in \mathbb{Z}_p$

let $\mathbf{F}(X) = a_{k-1}X^{k-1} + \cdots + a_2X^2 + a_1X + x$

give $\mathbf{F}(i)$ to party U_i.

The secret value is $\mathrm{x} = F(0)$.

sharing:



Shamir's secret sharing [2/2]

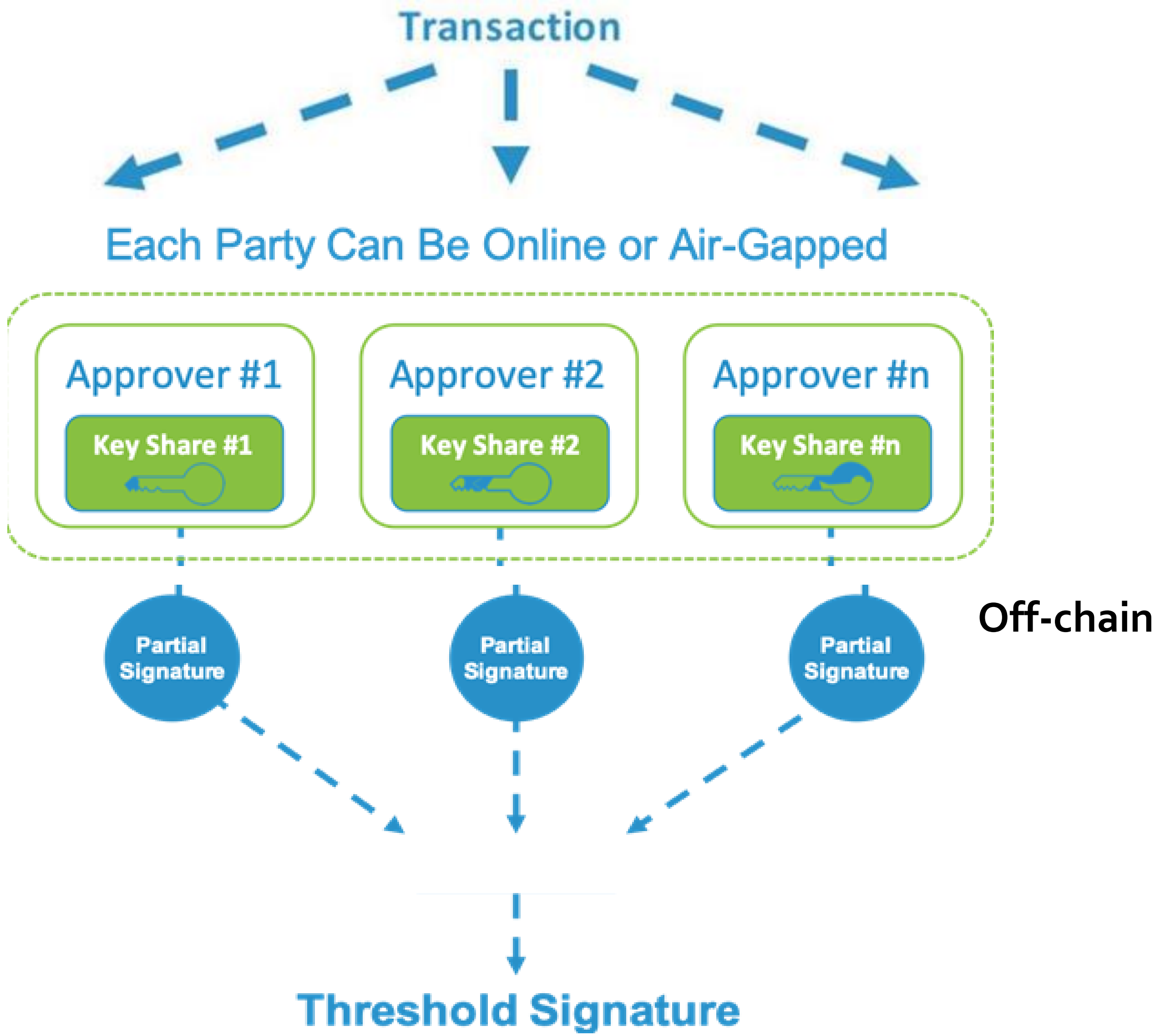
reconstruction:

Given $F(X_1),\dots,F(X_m)$ one can interpolate the polynomial F in point $F(0)$.

$$\lambda_k(X) = \prod_{\substack{i=1 \\ i \neq k}}^{t} \frac{X - X_i}{X_k - X_i} \pmod{p} \qquad k = 1, \dots, t$$

$$F(0) \equiv \sum_{k=1}^{t} F(k) \prod_{\substack{i=1 \\ i \neq k}}^{t} \frac{-X_i}{X_k - X_i} \pmod{p}$$

Threshold Signatures



Signature lengths

Goal: best existential forgery attack time $\geq 2^{128}$

algorithm	signature size
RSA	2048-3072 bits
EC-DSA	512 bits
Schnorr	384 bits
BLS	256 bits

Open problem: Practical and secure less than 256-bit signatures

BLS (Boneh-Lynn-Shacham) Signatures

Our first signature type

Let **G** be a cyclic group of prime order **q**

Efficient test τ to check if given $y, g, s, m \in G$

There exists $\mathbf{x} \in \mathbb{Z}_q$ such that $y = g^x$ and $s = M^x$

SK: x

PK: $y \leftarrow g^x$

On input a message m

Compute H(m) → M

$S = M^x$

m, S

Compute H(m) → M

Use test τ to check:

$y = g^x$ and $S = M^x$

DL and CDH is hard

DDH is easy

Gap DH problem !

D. Boneh; B. Lynn & H. Shacham (2004). "Short Signatures from the Weil Pairing". Journal of Cryptology. 17 (4)

Pairings Recap

$$e(P, Q)$$

$$a + b = b + a$$

$$e(P, Q + R) = e(P, Q) \times e(P, R))$$

$$a \times (b + c) = a \times b + b \times c$$

$$e(P + S, Q) = e(P, Q) \times e(S, Q))$$

$$(a \times c) + (b \times c) = (a + b) \times c$$

You can swap x for + an it should still work.
Lets try with: $e(x, y) = 2^{xy}$
Consider $x = 5$ and $y = 6$

$$e(5, 6) = 2^{5\times 6} = 2^{30}$$

$$e(5, 4 + 2) = e(5, 4) \times e(5, 2) = 2^{5\times 4} \times 2^{5\times 2} = 2^{30}$$

$$\forall a, b \in F_q^*, P \in G_1, Q \in G_2$$

$$e(aP, bQ) = e(P, Q)^{ab}$$

$$e(aP, bQ) = e(P, abQ) = e(abP, Q) = e(P, Q)^{ab}$$

BLS (Boneh-Lynn-Shacham) Signatures

EC - BLS

$$e(P,Q) \rightarrow n$$
$$e(xP,Q) = e(P,xQ)$$

$$\begin{aligned} e(G,S) &= e(P, M) \\ &= e(x \cdot G, M) \\ &= e(G, x \cdot M) \\ &= e(G, S)) \end{aligned}$$

$M \leftarrow H(m)$

Notice: Here we need a function that maps **H(m)** to a point on curve **M**

[BLS04, Sect 3.3]

PK: P $\leftarrow x\,.\,G$

SK: x

On input a message m

Compute **H(m) → *M***

S = $x \cdot M$

m, S

Compute **H(m) → *M***

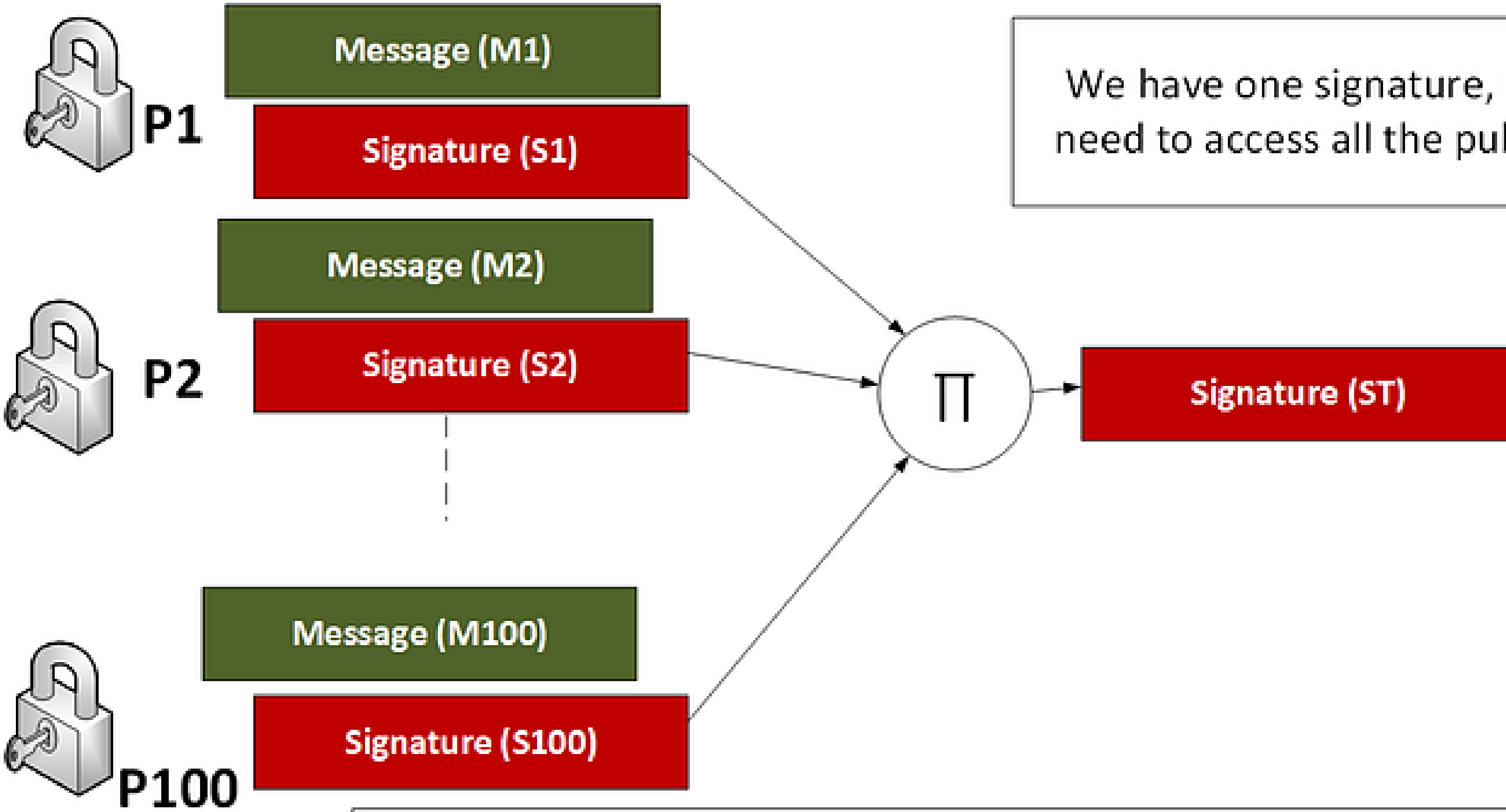
Use pairings to check:

$$e(G,S) = e(P,M)$$

D. Boneh; B. Lynn & H. Shacham (2004). "Short Signatures from the Weil Pairing". J.Cryptology. 17 (4)

Hash to curve (IETF) -> https://github.com/cfrg/draft-irtf-cfrg-hash-to-curve

BLS signature for Sign aggregation

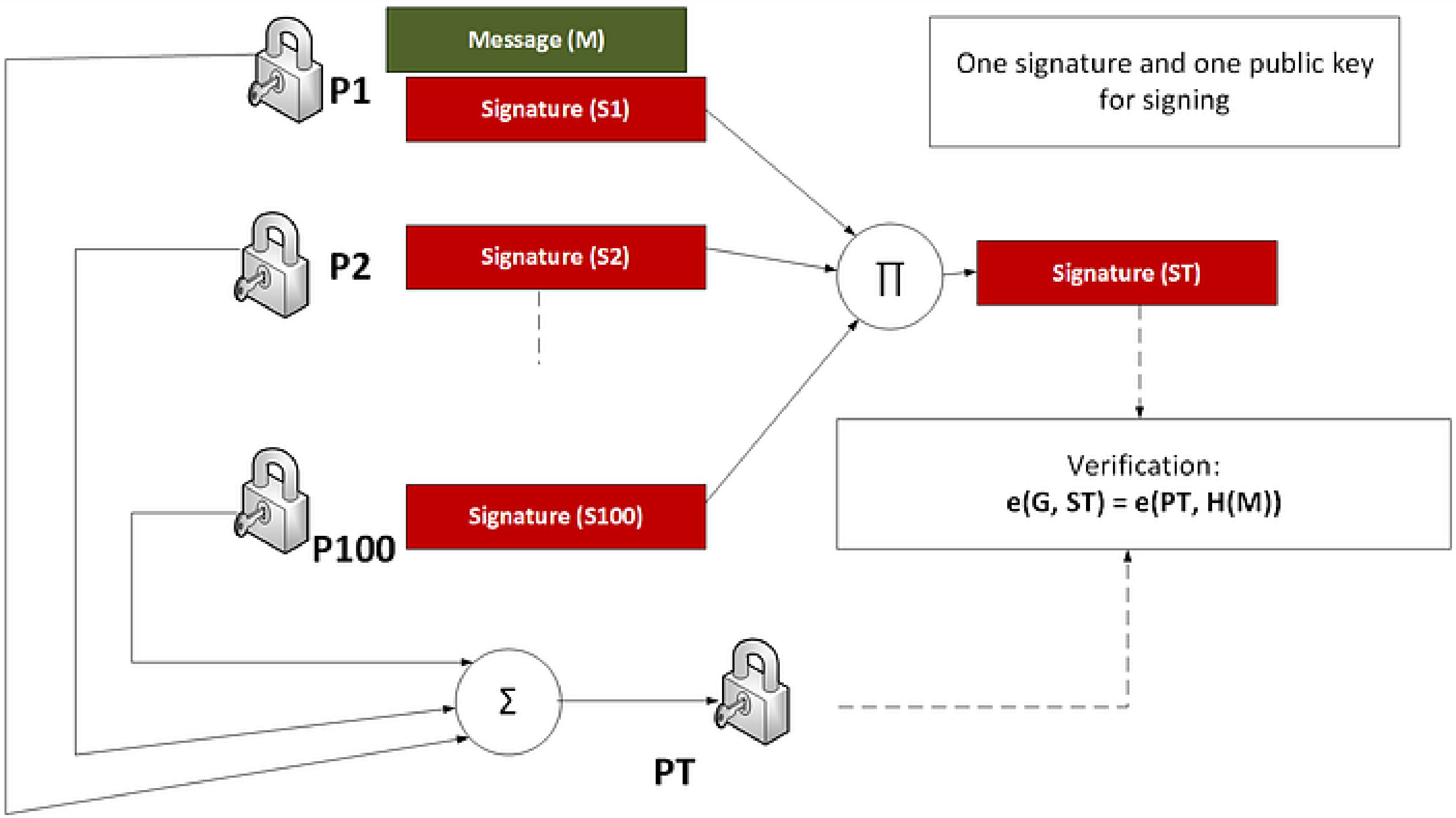


We have one signature, but still need to access all the public keys

Verification:

$$e(G, ST) = e(P1, H(m1)) \cdot e(P2, H(m2)) \cdot \ldots \cdot e(P100, H(m100))$$

BLS signature for multiSign



<u>Pairing friendly curves</u>

alt-bn128: used in Zchash and Ethereum’s Scalar Mult, point addition

BLS12-318: Ethereum 2.0 for signature aggregation and in some other chains

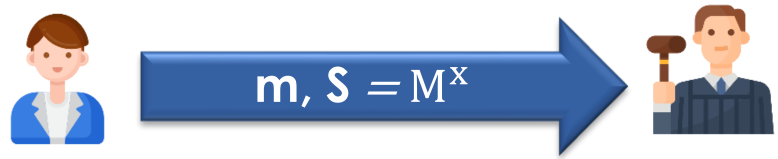
Secp256k1: used in Bitcoin, Ethereum and many others

Edwards25519: which is used in Cardano, Monero and many others

Threshold BLS Signatures

Threshold BLS

SK: x



PK: $P \leftarrow x \, . \, G$

$y = g^x$ and $S = M^x$

- A **dealer** shares the secret key x among **n** parties using Shamir SS.
  - Let $[x_1 \ \ldots \ x_n]$ be the shares
  - Remember there is a polynomial $\mathbf{F[X]}$ of degree **t** s.t. $\mathbf{F[0]} = x$ and $\mathbf{F[i]} = x_i$

- On input **m** every player outputs $S_i = M^{x_i}$
  - Given a set of **t+1** **partial signatures**
  - Then $S = \prod_{i \in S_i} S_i^{\lambda_{i,S}}$
    - Since $x = \sum_{i \in S} \lambda_{i,S} \, x_i$ and $S = M^x$

Interpolation in the exponent

- On input **m** every player outputs $S_i = x_i \cdot H(m)$
  - Given a set of **t+1** partial signatures
  - Then $S = \prod_{i \in S_i} \lambda_{i,S} \cdot S_i$
    - Since $x = \sum_{i \in S} \lambda_{i,S} \, x_i$ and $S = x \cdot H(m)$

EC - version

A. Boldyreva (2003), *Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme,* PKC 2003, LNCS- Volume 2567

Schnorr's Signature

Our second signature type

ZK-PK FS H

Let **G** be a cyclic group of prime order **q**

SK: x

PK: $y \leftarrow g^{x}$

On input a message m

Choose $\mathbf{k} \in_{\$} \mathbb{Z}_q$

Commit $\mathbf{R} = g^{k}$

Compute $\mathbf{c} = \mathbf{H}(\mathbf{m}, \mathbf{y}, \mathbf{R}) \in \mathbb{Z}_q$

Set $\mathbf{S} = \mathbf{k} + \mathbf{c}x \bmod q$

Output: **(R,S)**

m, R, S

Compute **M = H(m, y, R)**

Check : $g^{s} = Ry^{c}$

$$
\begin{aligned}
g^{s} &= Ry^{c} \\
g^{k+cx} &= g^{k}(g^{x})^{c} \\
g^{k+cx} &= g^{k}g^{cx}
\end{aligned}
$$

Schnorr, C.P. (1991) *Efficient signature generation by smart cards.* Journal of Cryptology 4, 161–174.

Threshold Schnorr Signature

Second example (Standard approach)

This only works for semi-honest adversary

SK: x PK: $y \leftarrow g^x$

On input a message m
Choose $\mathbf{k} \in_\$ \mathbb{Z}_q$
Compute $\mathbf{R} = g^k$
Compute $\mathbf{c} = \mathbf{H}(\mathbf{m}, \mathbf{y}, \mathbf{R}) \in \mathbb{Z}_q$
Set $\mathbf{S} = \mathbf{k} + \mathbf{c}x \bmod q$

m, R, S

Compute $M = H(m, y, R)$
Check : $g^s = Ry^c$

- A **dealer** shares the secret key **x** among **n** parties using Shamir SS.
  - Let $[x_1 \ \dots \ x_n]$ be the shares (polynomial $\mathbf{F[X]}$ of degree **t** s.t. $\mathbf{F[0] = x}$ and $\mathbf{F[i]} = \mathbf{x_i}$)
- A **dealer** shares the secret nonce **k** among **n** parties using Shamir SS.
  - Let $[k_1 \ \dots \ k_n]$ be the shares (polynomial $\mathbf{K[X]}$ of degree **t** s.t. $\mathbf{K[0] = k}$ and $\mathbf{K[i]} = k_i$)

Commitment is reconstructed

- On input **m** every player outputs $R_i = g^{k_i}$
  - Given a set of **S** of **t+1** partial nonces R_i we have that $R = \prod_{i \in S_i} R_i^{\lambda_{i,S}}$
  - Players can now compute $c = H(m, y, R)$ and $S_i = \mathbf{k_i} + \mathbf{c}x_i \bmod q$ partial signatures
  - Then $S = \sum_{i \in S} \lambda_{i,S} S_i$

What is special with k_i values

To Make TSS Robust

Robust version

SK: x PK: $y \leftarrow g^x$

On input a message m
Choose $\mathbf{k} \in_\$ \mathbb{Z}_q$
Compute $\mathbf{R} = g^k$
Compute $\mathbf{c} = \mathbf{H}(\mathbf{m}, \mathbf{y}, \mathbf{R}) \in \mathbb{Z}_q$
Set $\mathbf{S} = \mathbf{k} + \mathbf{c}x \bmod q$

m, R, S →

Compute $M = H(m, y, R)$
Check : $g^s = Ry^c$

- A **dealer** shares the secret key **x** among **n** parties using Shamir SS.
  - Let $[x_1 \ \ldots \ x_n]$ be the shares (polynomial $\mathbf{F[X]}$ of degree **t** s.t. $\mathbf{F[0] = x}$ and $\mathbf{F[i]} = \mathbf{x_i}$)
  - The dealer also publishes the commitments $\mathbf{PK_i} = \mathbf{y_i} = \mathbf{g^{x_i}}$

- A **dealer** shares the secret nonce **k** among **n** parties using Shamir SS.
  - Let $[k_1 \ \ldots \ k_n]$ be the shares (polynomial $\mathbf{K[X]}$ of degree **t** s.t. $\mathbf{K[0] = k}$ and $K[\mathbf{i}] = \boldsymbol{k_i}$)
  - The dealer also publishes the commitments $\mathbf{R} = \mathbf{g^k}$ and $\boldsymbol{R_i} = \mathbf{g^{k_i}}$

- On input **m** every player outputs $S_i = \mathbf{k_i} + \mathbf{c}x_i \bmod q$
  - A partial signature is checked by $g^{S_i} = R_i y_i^c$
  - Only sellect a set of **t+1** correct **partial signatures**
  - Then compute $S = \sum_{i \in S} \lambda_{i,S} S_i$

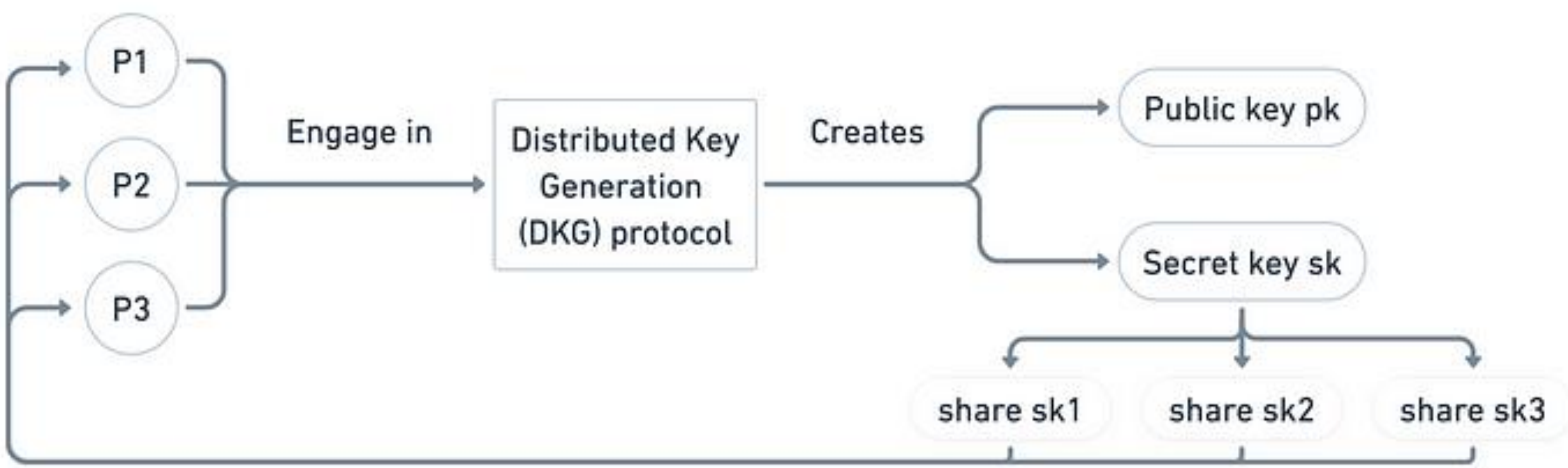
Bottleneck → Dealer?

- We have assumed that there is a **dealer** that prepares and shares some data to the parties.
  - Is this a single point of failure?
    - **YES!!**
- Which treshold signature is worse that we have seen so far in terms of dealer dependency?
  - Threshold BLS or Threshols Schnorr
- A dealer who shares the secret key x is a SPF at the begining
- Only considering some certain cases this dealer can be accepted
  - By setting highly secure trusted set-up
  - Destroying all residual data
- A dealer who shares the secret nonce **k** is equivalent to knowledge of **x** everytime a signature is issued
- Can we do without a dealer
    - YES
      - **DKG: Distrubuted Key Generation** (needs to be used to generate nonce as well)
      - next slides…

What properties do we need

- The n players should jointly generate a sharing of secret key x
  - Let $[x_1 \; \dots \; x_n]$ be the private secret key shares
  - The public key $\mathbf{PK} = y = g^x$
  - The partial public keys $\mathbf{PK_i} = y_i = g^{x_i}$
- This protocol is repeated for each signature to generate the nonce k
  - Let $[k_1 \; \dots \; k_n]$ be the private random shares,

  the public nonce $R = g^k$ public nonce and the partial nonces $R_i = g^{k_i}$



What properties do we need

- With a simulatable DKG we can construct Threshold Signatures for discrete-log based schemes such as BLS and Schnorr
  - Honest majority with robustness
    - **Joint-Pedersen DKG**
  - Dishonest Majority with abort
    - **Committed Pedersen DKG**

- Proof follows a simulation argument
  - If you can forge in the threshold setting you can also forge in the centralised setting

What about (EC)DSA: The Digital Signature Algorithm

Let **G** be a cyclic group of prime order **q**

SK: x PK: y $\leftarrow g^x$

On input a message M

Choose **k** $\in_\$ \mathbb{Z}_q$

Compute **R** $= g^{k^{-1}}$

Set **S** $= \mathbf{k}(\mathbf{m} + \boldsymbol{xr})\ mod\ q$

- **r = H(R)** and **m =H(M)** $\in \mathbb{Z}_q$

Signature **(R,S)**

M, R, S

Compute **r = H(R), m=H(M)**

Check if: $R^S = g^m y^r$

$$
\begin{aligned}
R^S &= g^m y^r \\
R^{k(m+xr)} &= g^m (g^x)^r \\
g^{k^{-1}k(m+xr)} &= g^m g^{xr} \\
g^{m+xr} &= g^{m+xr}
\end{aligned}
$$

David W Kravitz. Digital signature algorithm, 1993. US Patent 5,231,668 (adopted by NIST)

Schnorr vs (EC)DSA

Schnorr Signature

- On input a message M
  - Choose $\mathbf{k} \in_{\$} \mathbb{Z}_q$
  - Compute $\mathbf{R} = g^k$
  - Compute $\mathbf{c} = \mathbf{H}(\mathbf{M}, \mathbf{y}, \mathbf{R}) \in \mathbb{Z}_q$
  - Set $\mathbf{S} = \mathbf{k} + \mathbf{c}x \bmod q$
  - Signature **(R,S)**

Linear combination of k and x secret values

(EC)DSA

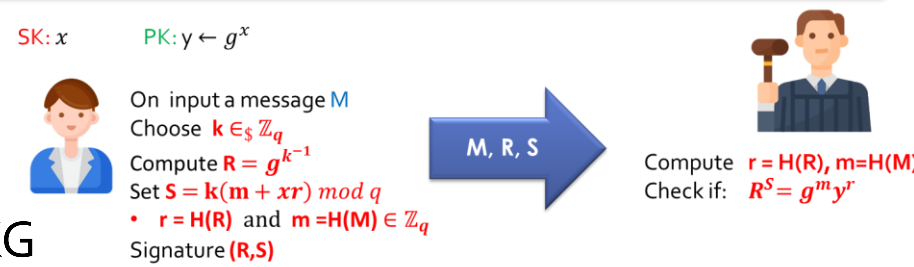
- On input a message M
  - Choose $\mathbf{k} \in_{\$} \mathbb{Z}_q$
  - Compute $\mathbf{R} = g^{k^{-1}}$
  - $\mathbf{r} = \mathbf{H}(\mathbf{R})$ and $\mathbf{m} = \mathbf{H}(\mathbf{M}) \in \mathbb{Z}_q$
  - Set $\mathbf{S} = \mathbf{k}(\mathbf{m} + x\mathbf{r}) \bmod q$
  - Signature **(R,S)**

1) Computing Inversion

2) Multiplication of two secret shared values

Robust Threshold (EC)DSA

1) Computing Inversion



- Players perform two Joint-Pedersen DKG
  - Let k, a be the random values generated
  - Only for a the Feldman VSS phase is performed, so the value $A = g^a$ is public
- Players reconstruct the value $b = ka$
  - By broadcasting the product shares
    - Requires randomisation with a 0-polynomial of degree 2t (which has a free term k)

$$g^{ac} = g^{a(ka)^{-1}} = g^{k^{-1}}$$

- The players $c = b^{-1} \bmod q$ and compute $R = A^c = g^{k^{-1}}$ (now it is public)
  - The players have shares of $k = [k_1, \ldots, k_n]$

Beaver, D. (1991). Efficient Multiparty Protocols Using Circuit Randomization - CRYPTO '91

Gennaro, R., Goldfeder, S., Narayanan, A. (2016). Threshold-Optimal DSA/ECDSA Signatures and an Application to Bitcoin Wallet Security. ACNS 2016

Robust Threshold (EC)DSA

2) Multiplication of two secrets shared additively

- Assume n players have additive shares of secrets $\boldsymbol{a}, \boldsymbol{b}$
  - $\boldsymbol{a} = \boldsymbol{a_1} + \cdots + \boldsymbol{a_n}$ and $\boldsymbol{b} = \boldsymbol{b_1} + \cdots + \boldsymbol{b_n}$
  - Player i hold a_i and b_i

- The parties want to compute an additive sharing of ***c =ab***
  - Note that $\boldsymbol{c} = \sum_{i,j} \boldsymbol{a_i b_j}$
  - If parties i and j could turn $a_i b_j$ into two values d_{ij} and e_{ij} s.t.
    - $\boldsymbol{d_{ij}} + \boldsymbol{e_{ij}} = \boldsymbol{a_i b_j}$
  - Then Player i could set c_i to
    - $a_i b_i + \sum_i d_{ij} + \sum_i e_{ji}$
  - $\boldsymbol{c} = \boldsymbol{c_1} + \cdots + \boldsymbol{c_n}$

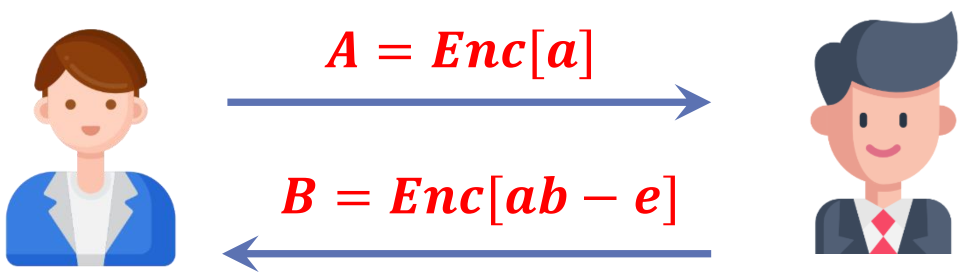
O. Goldreich, S. Micali, and A. Wigderson. (1987) How to play any mental game or a completeness theorem for protocols with honest majority. STOC'87

MtA - Multiplicative to Additive Shares

- An **MtA** protocol let two players Alice and Bob
  - Parties hold secret values $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{Z}_q$ respectively
  - To turn them into secret $\boldsymbol{d}, \boldsymbol{e} \in \mathbb{Z}_q$ s.t.
  - $\boldsymbol{d} + \boldsymbol{e} = \boldsymbol{ab} \ mod\ q$

- Let ***Enc*** be an additively Homomorphic Encryption scheme
  - with message space and homomorphism over $\mathbb{Z}_q$

Knows $\boldsymbol{Dec}$



$\boldsymbol{A} = \boldsymbol{Enc}[\boldsymbol{a}]$

$\boldsymbol{B} = \boldsymbol{Enc}[\boldsymbol{ab} - \boldsymbol{e}]$

Picks $\boldsymbol{e} \in_{\$} \mathbb{Z}_q$
Uses homomorphism to compute $\boldsymbol{B}$

$\boldsymbol{d} \leftarrow \boldsymbol{Dec}[\boldsymbol{B}]$

$\boldsymbol{d} + \boldsymbol{e} = \boldsymbol{ab}$

What HE scheme can be used?

How about Paillier?

- In normal case **q** is determined by **DSA** parameters
  - **Enc** with message space and homomorphism over $\mathbb{Z}_q$
- What about Paillier?
  - Homomorphism is over Z_N where is a modules (as RSA modulus)
  - Parties need to add a **Range ZK-proof** that their values are ``small``
    - Prevent reduction mod N
    - This is important to ensure both privacy and correctness

Knows $\boldsymbol{Dec}$

$\boldsymbol{A = Enc[a]}$+ZKP[a small] →

← $\boldsymbol{B = Enc[ab - e]}$ +ZKP[b,e small]

Picks $\boldsymbol{e} \in_{\$} \mathbb{Z}_q$
Uses homomorphism to compute $\boldsymbol{B}$

$\boldsymbol{d \leftarrow Dec[B]}$

$\boldsymbol{d + e = ab}$

+ Committed Pedersen DKG

R Canetti, R Gennaro, S Goldfeder, N Makriyannis, U. Peled (2020) *UC non-interactive, proactive, threshold ECDSA with identifiable aborts* - ACM SIGSAC, 2020

FROST: Flexible Round Oprtimised Schnorr Threshold Signature

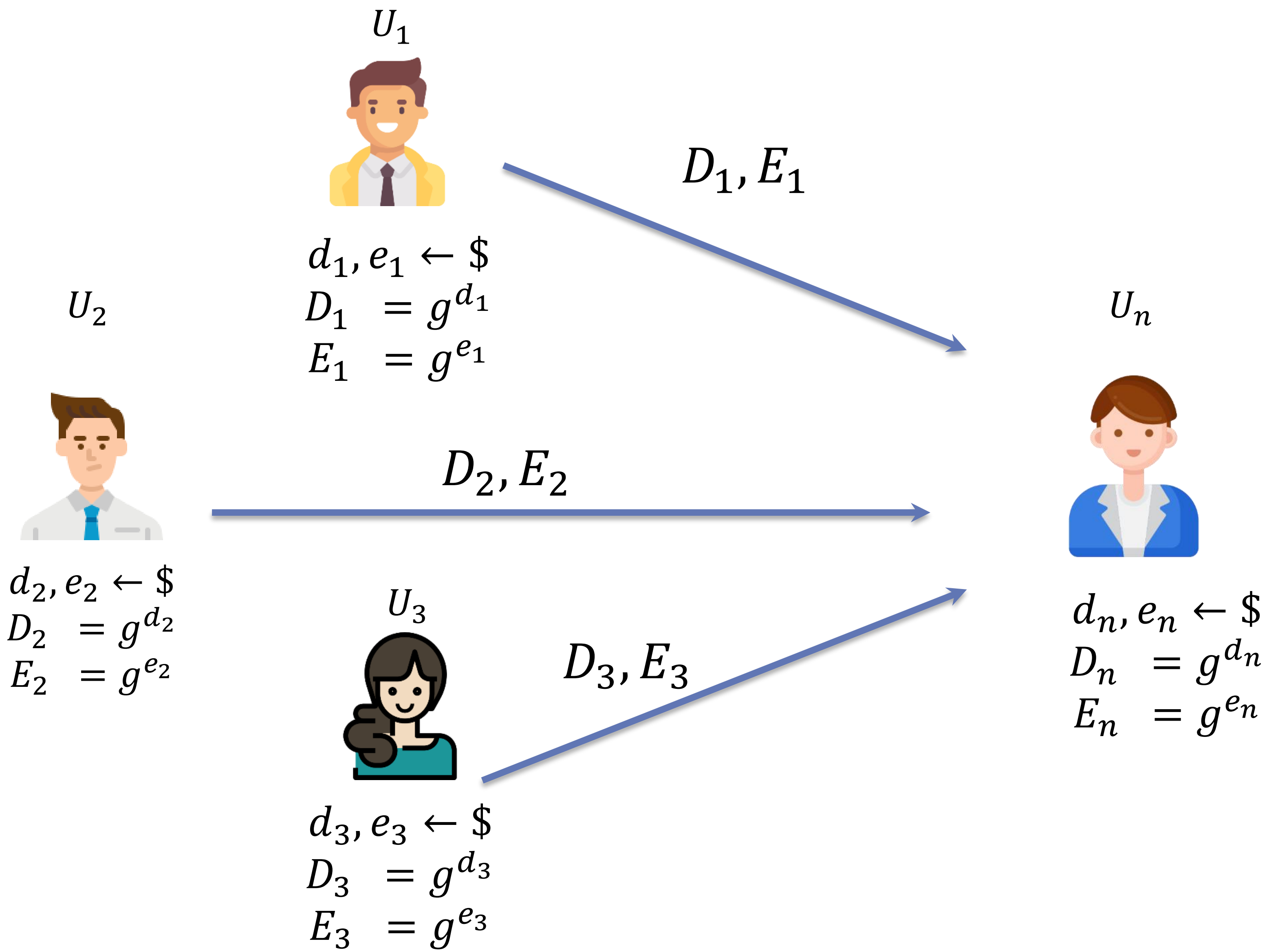
State-of-the art

- Main improvements over the Schnorr TSS
  - Assumes dishonest majority
  - If there are t honest parties, will a valid signature be formed even if the presence of a malicious party?
    - Even if robustness is not fully satisfied dishonest parties can be kicked out one by one
      - New print to make FROST robust -> ROAST Ruffing et. al.
  - Woks securely in the parallel setting and with dishonest majority (Secure against Drijvers attack)
  - Output is identcal to single-party Schnorr.
  - **2 Round signing where the first can be done in processing Round**
  - Some standatisains (CFRG).

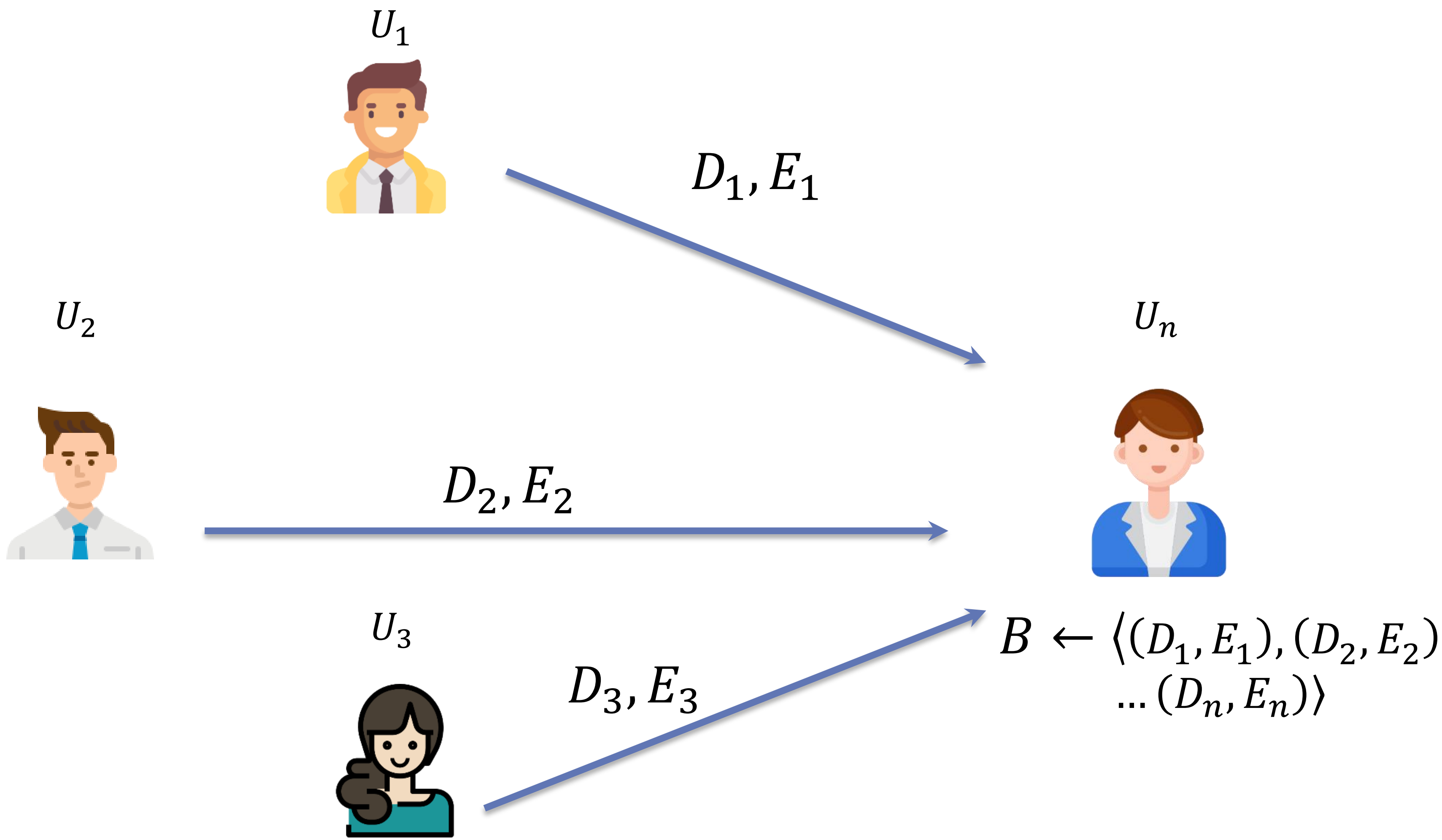
Komlo, C., Goldberg, I. (2021). FROST: Flexible Round-Optimized Schnorr Threshold Signatures. SAC 2020

https://www.ietf.org/archive/id/draft-irtf-cfrg-frost-03.html#RFC8032

FROST


U_1
$d_1, e_1 \leftarrow \$$
$D_1 = g^{d_1}$
$E_1 = g^{e_1}$
D_1, E_1
U_2
$d_2, e_2 \leftarrow \$$
$D_2 = g^{d_2}$
$E_2 = g^{e_2}$
D_2, E_2
U_3
$d_3, e_3 \leftarrow \$$
$D_3 = g^{d_3}$
$E_3 = g^{e_3}$
D_3, E_3
U_n
$d_n, e_n \leftarrow \$$
$D_n = g^{d_n}$
$E_n = g^{e_n}$

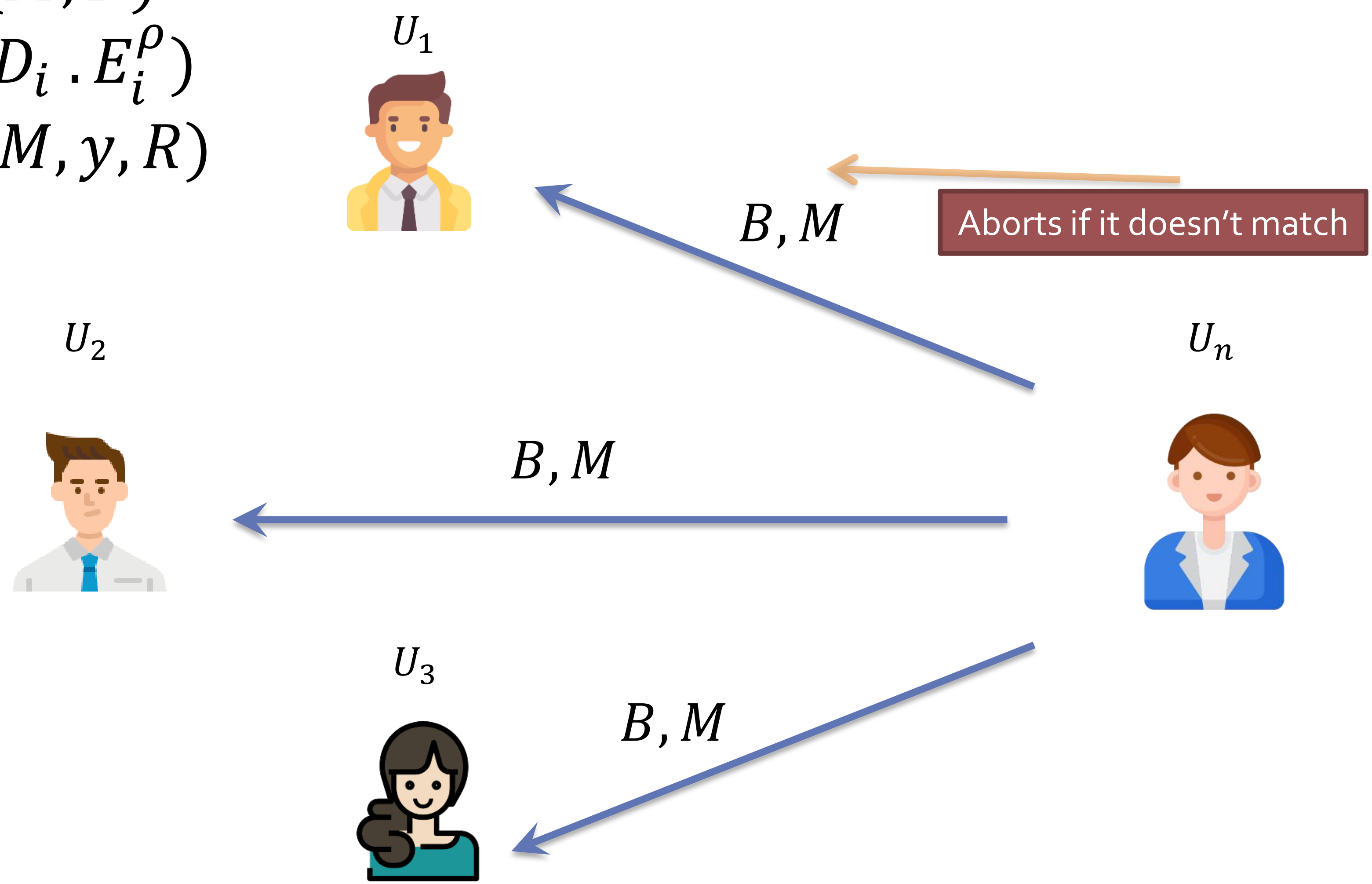
FROST


U_1
D_1, E_1
U_2
D_2, E_2
U_n
U_3
D_3, E_3
$B \leftarrow \langle (D_1, E_1), (D_2, E_2) \ldots (D_n, E_n) \rangle$

FROST

SK: x PK: $y \leftarrow g^x$

$$\rho \leftarrow H_1(M, B)$$
$$R \leftarrow \prod(D_i \,.\, E_i^{\rho})$$
$$c \leftarrow H_2(M, y, R)$$

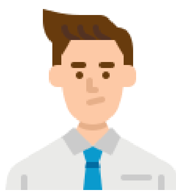


FROST

U_1

$$s_1 = d_1 + \rho e_1 + \lambda_1 x_1 c$$

U_2



$$s_2 = d_2 + \rho e_2 + \lambda_2 x_2 c$$

U_3

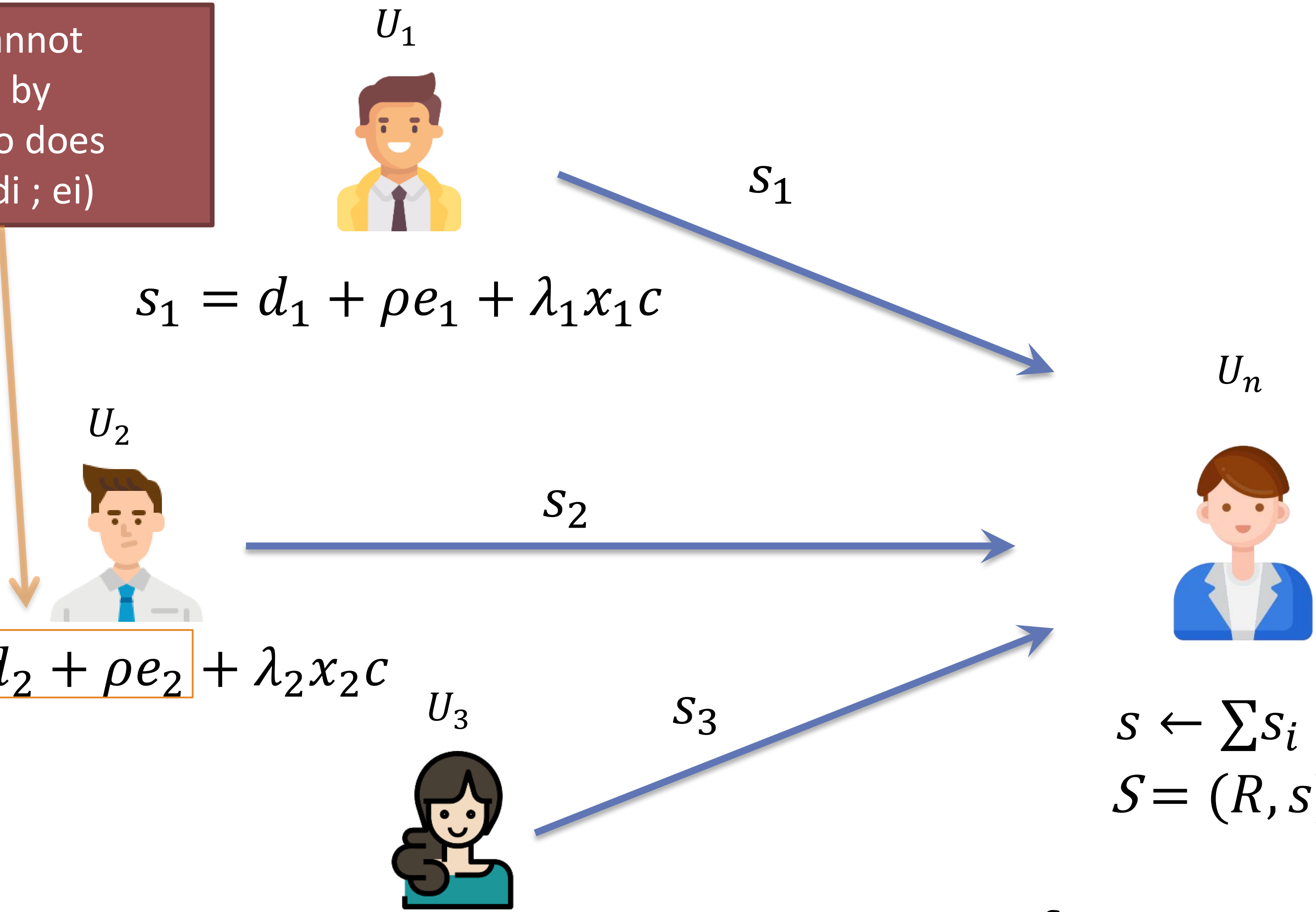
$$s_3 = d_3 + \rho e_3 + \lambda_3 x_3 c$$

U_n

$$s_n = d_n + \rho e_n + \lambda_4 x_4 c$$

FROST

This step cannot be inverted by anyone who does not know (di ; ei)



$$s_1 = d_1 + \rho e_1 + \lambda_1 x_1 c$$

$$s_2 = \boxed{d_2 + \rho e_2} + \lambda_2 x_2 c$$

$$s_3 = d_3 + \rho e_3 + \lambda_3 x_3 c$$

$$s \leftarrow \sum s_i$$

$$S = (R, s)$$

$$g^{s_i} = R_i \cdot y_i^{c.\lambda_i}$$

More references on Threshold (EC)DSA with abort

Dishonest Majority

- Similar techniques with [CGGMP20] based on ECDSA
  - Lindell, Y., Nof, A.(2018) Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody. ACM-CCS 2018
- OT-based MtA protocol (Uses only ECDSA assumption but bit-by-bit operations)
  - Doerner, J., Kondi, Y., Lee, E., Shelat, A. (2018) Secure two-party threshold ECDSA from ECDSA assumptions. In: 2018 IEEE – S&P'18
- Using class groups in the MtA protocol (no range proofs)
  - G. Castagnos, D. Catalano, F. Laguillaumie, F. Savasta, I. Tucker (2023), Bandwidth-efficient threshold EC-DSA revisited: Theoretical Computer Science, Volume 939, Pages 78-104
- Using MPC techniques
  - D. Abram, A. Nof, C. Orlandi, P. Scholl and O. Shlomovits, (2022) "Low-Bandwidth Threshold ECDSA via Pseudorandom Correlation Generators,"IEEE Symposium on Security and Privacy (S&P)
  - Dalskov, A., Orlandi, C., Keller, M., Shrishak, K., Shulman, H. (2020). Securing DNSSEC Keys via Threshold ECDSA from Generic MPC, ESORICS 2020
- Two-party case
  - MacKenzie, P.D., Reiter, M.K.(2004) Two-party generation of DSA signatures. Int. J. Inf. Sec. 2(3-4), 218–239
  - Lindell, Y. (2017) Fast secure two-party ECDSA signing. Advances in Cryptology – CRYPTO'2017

More references on Threshold (EC)DSA with abort

Honest Majority

- Assumes n>2t+1 but also aborts
- Trade-off is better efficiency and Round complexity
- Also no need for a relaible broadcast channel (required by a robust and fair protocol)

Damgård, I., Jakobsen, T.P., Nielsen, J.B., Pagter, J.I., Østergaard, M.B. (2020). Fast Threshold ECDSA with Honest Majority. Security and Cryptography for Networks. SCN 2020

Questions?

Dr Muhammed Ali Bingol

muhammed.bingol@dmu.ac.uk

‘‘The **scientist** is not a person who gives the right answers, he's one who **asks** the **right questions**.’’ *Claude Levi-Strauss*