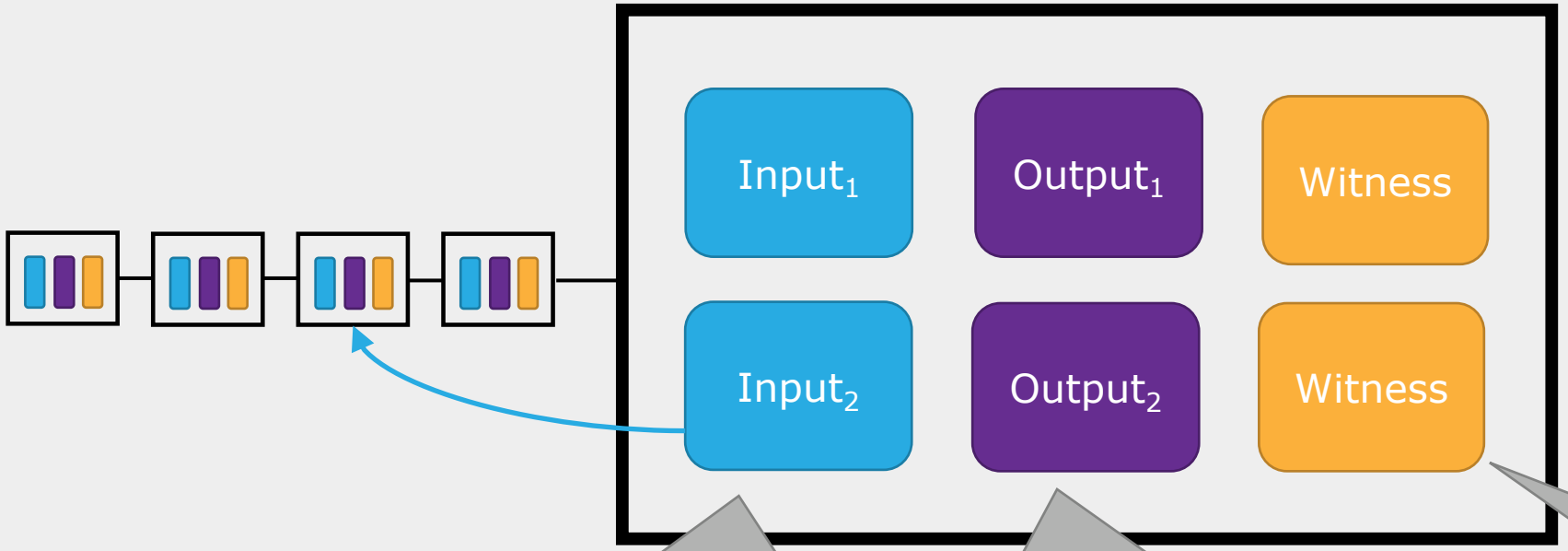
Compact Multi-Signatures for Smaller Blockchains

Dan Boneh[1], Manu Drijvers[2], Gregory Neven[2]

[1] Stanford University
[2] DFINITY

Bitcoin Blockchain and transactions



Pointer to previous output with

$addr_{in}$ = H(pk)
$amount_{in}$ = 1 BTC

recipient address & amount

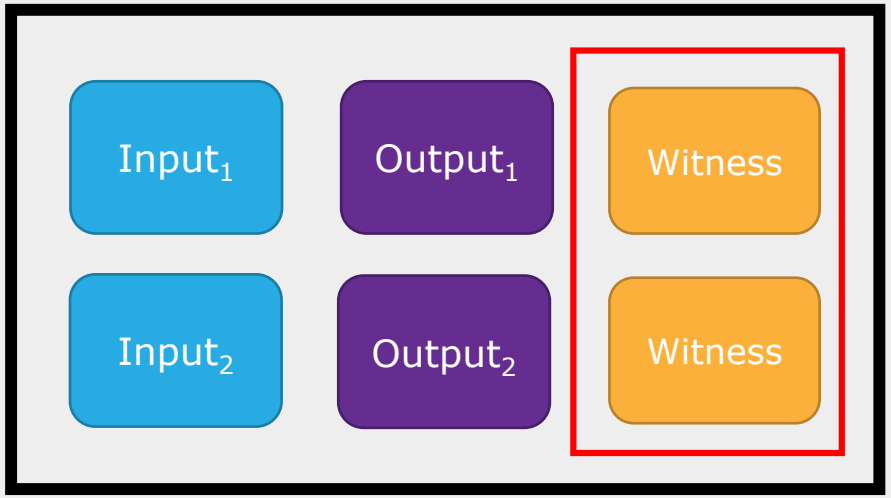
$addr_{out}$ = H(pk')
$amount_{out}$ = 1 BTC

Witness data for all transactions

pk, σ under pk

Saving space is important

- Larger transactions mean higher network and storage requirements
  - Current Bitcoin blockchain is almost 200 GB
- Block size is limited
  - Limits transaction throughput
  - Smaller transactions can mean higher throughput
- Goal: minimize total witness size using multisignatures



Multi-Signature Schemes



$(sk_1, pk_1) \leftarrow \mathrm{KGen}(1^\kappa)$ $(sk_2, pk_2) \leftarrow \mathrm{KGen}(1^\kappa)$ $(sk_3, pk_3) \leftarrow \mathrm{KGen}(1^\kappa)$

$\mathrm{Sign}(pk_1, pk_2, pk_3, sk_1, m) \leftrightarrow \mathrm{Sign}(pk_1, pk_2, pk_3, sk_2, m) \leftrightarrow \mathrm{Sign}(pk_1, pk_2, pk_3, sk_3, m)$

$\hookrightarrow \sigma$ $\hookrightarrow \sigma$ $\hookrightarrow \sigma$

- $\mathrm{Ver}(\{pk_1, pk_2, pk_3\}, \sigma, m) = 1/0$
- **Every** signer must agree to signing m
- Key aggregation: $apk = \mathrm{KAgg}(\{pk_1, pk_2, pk_3\})$
  - $\mathrm{Ver}(apk, \sigma, m) = 1/0$

Recap: Boneh-Lynn-Shacham signatures

Let $G_1 = \langle g_1 \rangle, G_2 = \langle g_2 \rangle, G_t = \langle g_t \rangle,$ with bilinear pairing e

- **KGen**: $pk = {g_2}^{sk}$
- **Sign(sk, m)**: $\sigma = H(m)^{sk}$
- **Verify(pk, σ, m)**: $e(\sigma, g_2) = e(H(m), pk)$

Naïve BLS Multi-signatures

Let $G_1 = \langle g_1 \rangle, G_2 = \langle g_2 \rangle, G_t = \langle g_t \rangle,$ with bilinear pairing e

- **KGen**: $pk_i = g_2^{sk_i}$
- **KAgg(pk$_1$, …, pk$_n$)**: $apk = \prod pk_i$
- **Sign(pk$_1$, …, pk$_n$, sk$_i$, m)**: $s_i = H(m)^{sk_i},\ \sigma = \prod s_i$
- **Verify(apk, σ, m)**: $e(\sigma, g_2) = e(H(m), apk)$

Rogue-Key Attack: Adversary chooses $pk = \frac{g_2^{sk}}{pk^*}$,
Adversary can sign for $\{pk, pk^*\}$ by setting $\sigma = H(m)^{sk}$

Can be mitigated using “proofs-of-possession” [RY07]

New BLS Multi-signatures without PoPs

Let $G_1 = \langle g_1 \rangle, G_2 = \langle g_2 \rangle, G_t = \langle g_t \rangle,$ with bilinear pairing e

- **KGen**: $pk_i = g_2^{sk_i}$
- **KAgg(pk$_1$, …, pk$_n$)**: $apk = \prod pk_i^{a_i}$, with $a_i = H_1(pk_i, \{pk_1, \ldots, pk_n\})$

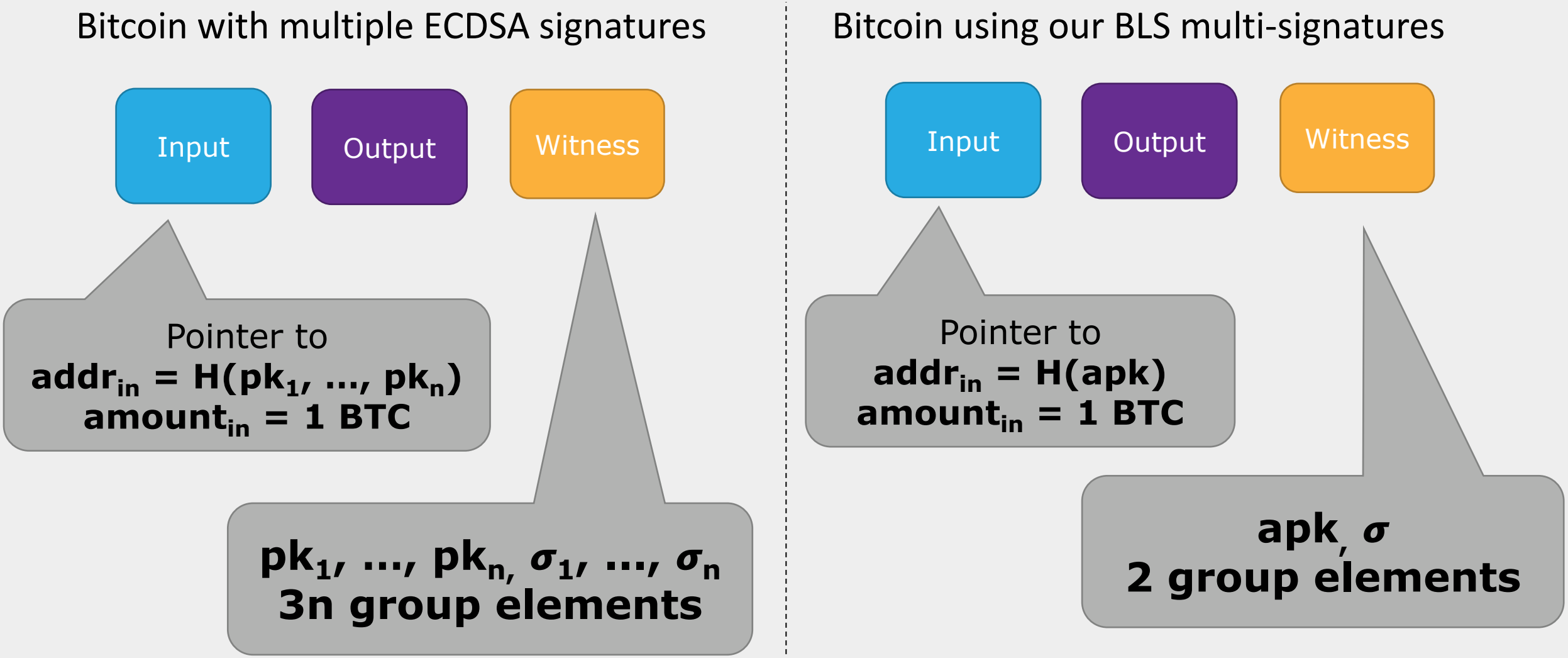
- **Sign(pk$_1$, …, pk$_n$, sk$_i$, m)**: $s_i = H_0(m)^{sk_i}, \quad \sigma = \prod s_i^{a_i}$

- **Verify(apk, σ, m)**: $e(\sigma, g_2) = e(H_0(m), apk)$

Uses trick from [Maxwell-PSW18]

Thm: secure multi-signature scheme under co-CDH in ROM

Bitcoin Multisig address



Aggregatable Multi-Signatures

Extend multi-signature definition with two additional algorithms

- **SigAgg({apk_i, m_i, σ_i}):** Aggregate a set of multi-signatures into a single object
- **AggVerify({apk_i, m_i}, Σ):** Verify that the aggregate multi-signature

For our BLS multisignature scheme

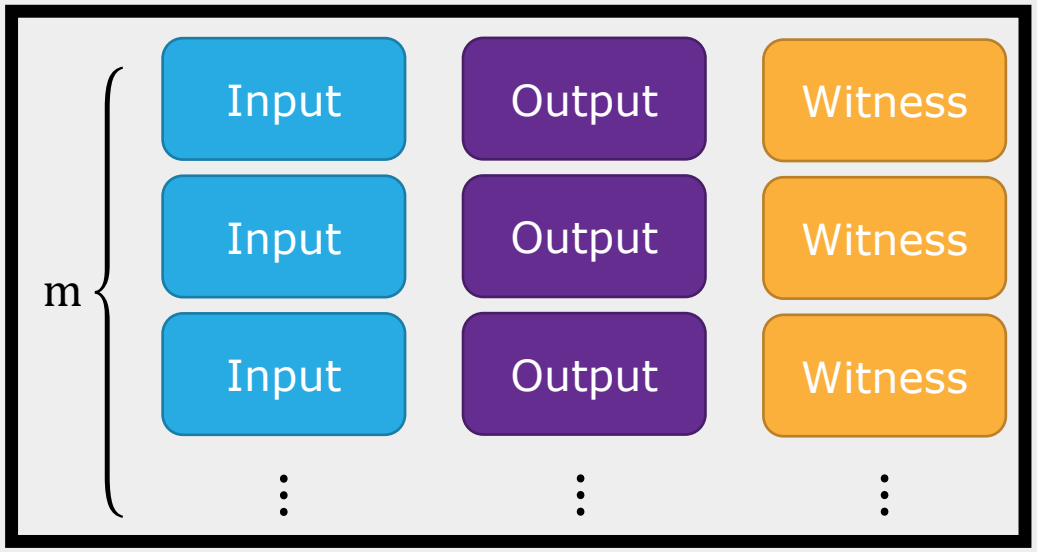
- **Sign(pk_1 , …, pk_n , sk_i, m)**: $s_i = H_0(apk, m)^{sk_i}, \qquad \sigma = \prod s_i^{a_i}$
- **SigAgg({apk_i, m_i, σ_i}):** $\Sigma = \prod \sigma_i$
- **AggVerify({apk_i, m_i}, Σ):** $e(\Sigma, g_2) = \prod e(H_0(m_i), apk_i)$

Thm: secure aggregatable multisignature scheme under ψ-co-CDH in ROM

Aggregatable Multi-Signatures in Bitcoin

Block of *m* transactions, each spending from an *n*-multisig address

Bitcoin with multiple ECDSA signatures

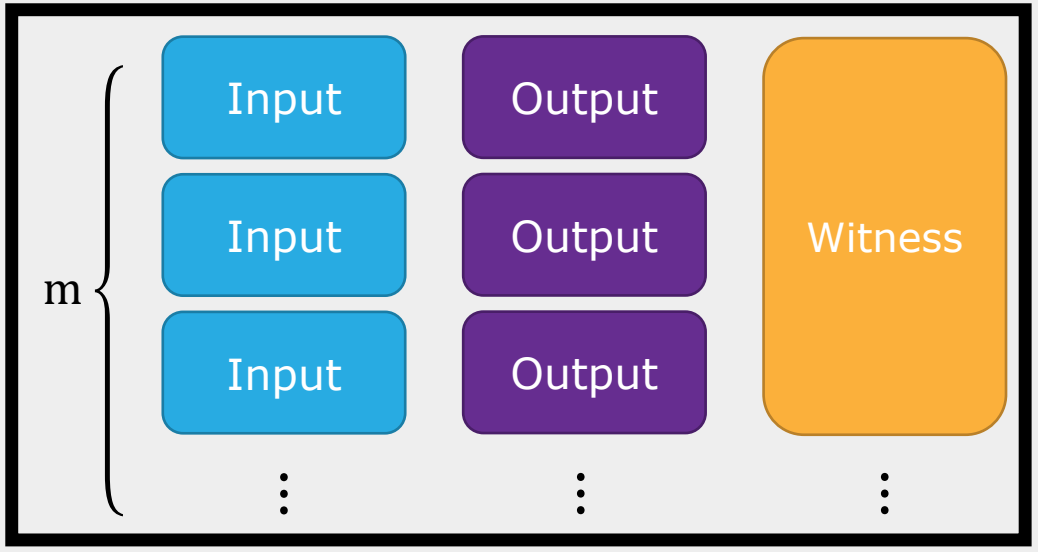


Combined witness contains

- $n \cdot m$ public keys
- $n \cdot m$ signatures

1296 KB

Bitcoin using aggregatable multi-signatures



Combined witness contains

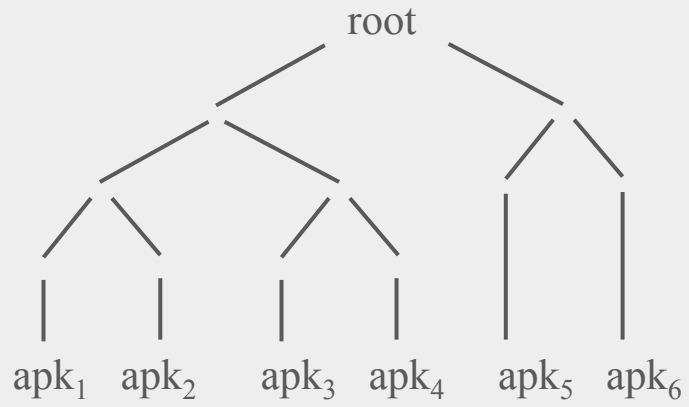
- m (aggregate) public keys
- 1 aggregate multi-signature

216 KB

t-out-of-n wallets

- Multi-signatures always require n-out-of-n, what about other policies?
- Typical threshold wallets have $\text{addr} = H(pk_1, \ldots, pk_n, t)$
  - Need to reveal n keys and t signatures

- For small $\binom{n}{t}$, use multi-signatures
  - Exhaustively list *apk* of all t-size subsets



Can we handle arbitrary t, n?

Yes! Using a new Accountable Subgroup Multi-signature (ASM)

- Aggregate public key $apk \in G_2$
- Any subset $S = [1, 1, 0, \ldots]$ can sign in an accountable way
- Signature $\sigma \in G_1 \times G_2$
- **Thm**: under ψ-co-CDH in ROM

- t-out-of-n Bitcoin transaction
  - Reveal apk, σ, S
  - Almost constant: 3 group elements + n bits

Our ASM Scheme

Alice $pk_1 = g_2^{sk_1}$

Bob $pk_2 = g_2^{sk_2}$

Charlie $pk_3 = g_2^{sk_3}$

Key aggregation: $apk = \prod pk_i^{a_i} = g_2^{ask}$

Membership keys:

$mk_1 = H_2(apk, 1)^{ask}$ ⟷ $mk_2 = H_2(apk, 2)^{ask}$ ⟷ $mk_3 = H_2(apk, 3)^{ask}$

Sign:

$s_1 = H_0(apk, m)^{sk_1} \cdot mk_1$ $\quad$ $s_2 = H_0(apk, m)^{sk_2} \cdot mk_2$

Combine: $(k = pk_1 \cdot pk_2, s = s_1 \cdot s_2)$

3 pairings

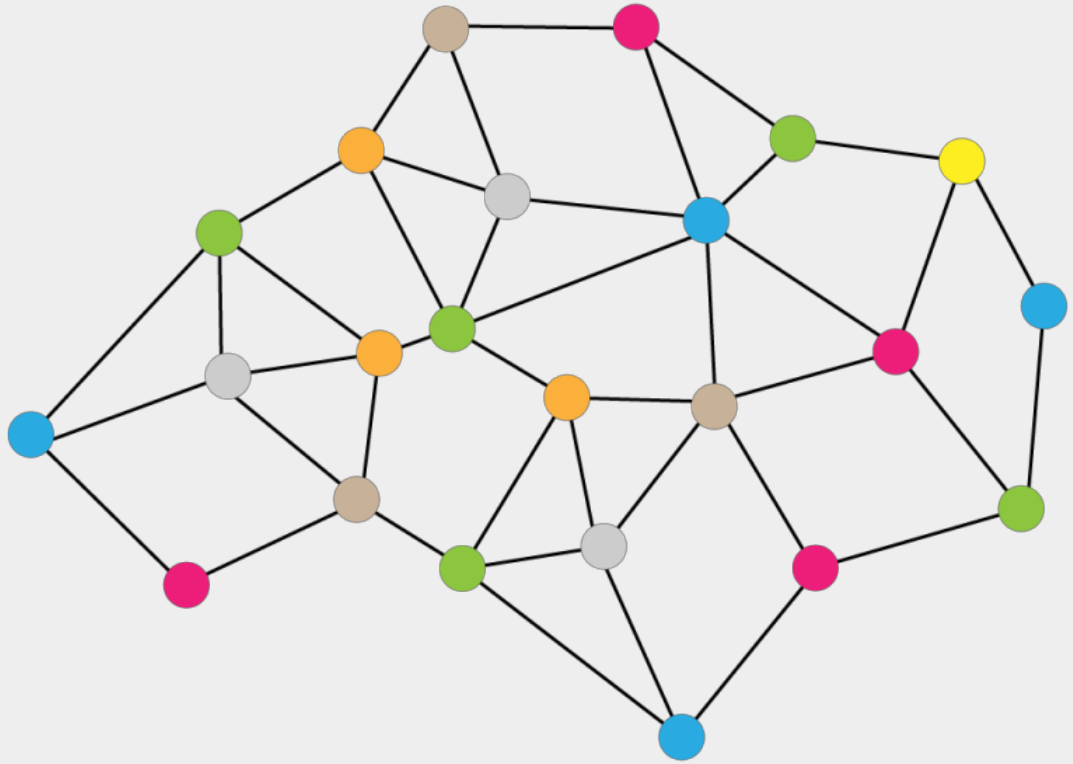
Verify(apk, S=[1,1,0], (k, s)) : $e(s, g_1) = e(\prod_{i \in S} H_2(apk, i), apk) \cdot e(H_2(apk, m), k)$

Conclusion

- BLS multi-signatures without PoPs
  - Support key aggregation
  - Support aggregation of multi-signatures
- Accountable Subgroup Multi-signatures
  - Key aggregation
  - Any subgroup can create constant size accountable multi-signature
  - Supports partial aggregation
- Schnorr multi-signatures without PoPs
- All schemes with PoPs

Thanks!

ia.cr/2018/483



Can we handle arbitrary t, n?

Yes! Using a new Accountable Subgroup Multi-signature (ASM)

- Aggregate public key $apk \in G_2$
- Any subset $S = [1, 1, 0, \ldots]$ can sign in an accountable way
- Signature $\sigma \in G_1 \times G_2$
- **Thm**: under ψ-co-CDH in ROM