COMP4913 Capstone Project
Blockchain Unleashed: A Secure E-Voting Application

Student Name: Yiu Kam Wing

Student ID: 20028987D

Why use blockchain?

Traditional E-Voting Architecture

Traditional E-Voting Architecture

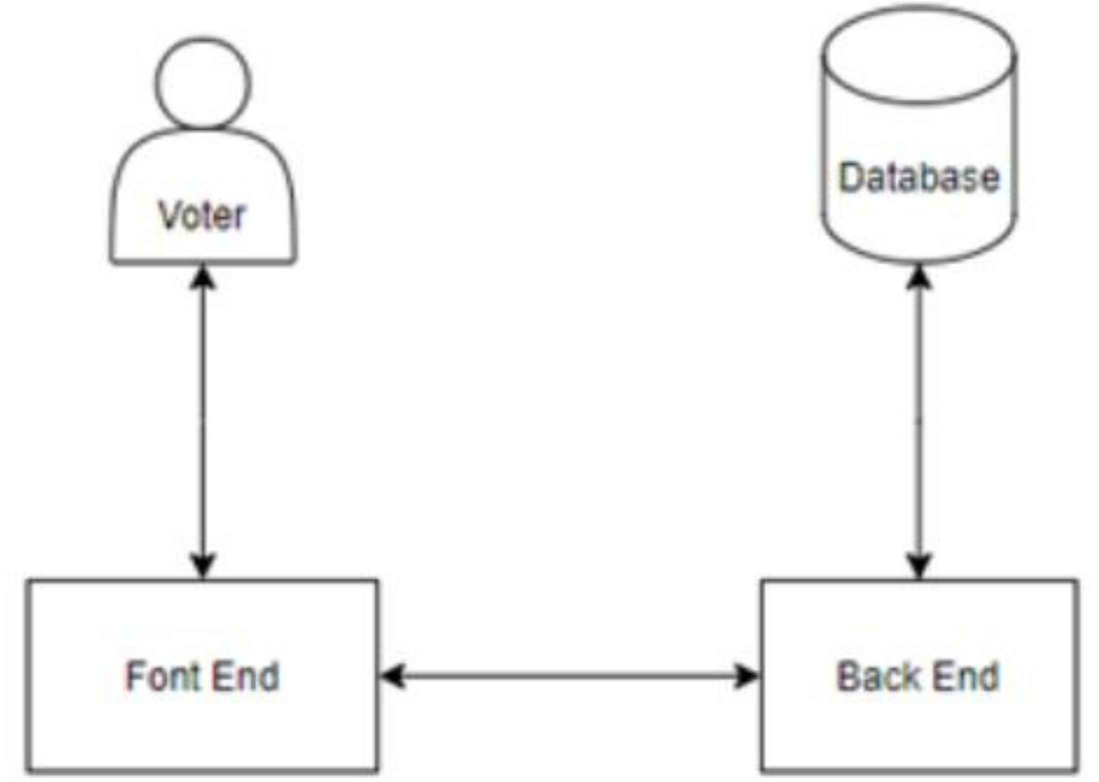


Figure 1.1: Traditional E-Voting Architecture

Problems of Traditional E-Voting Architecture

- Opacity
  - organizations or authorities have complete control over the database and system
- Single point of failure
  - The database and the server can be compromised by the hacker without anyone knowing.

X: the credibility of the voting result

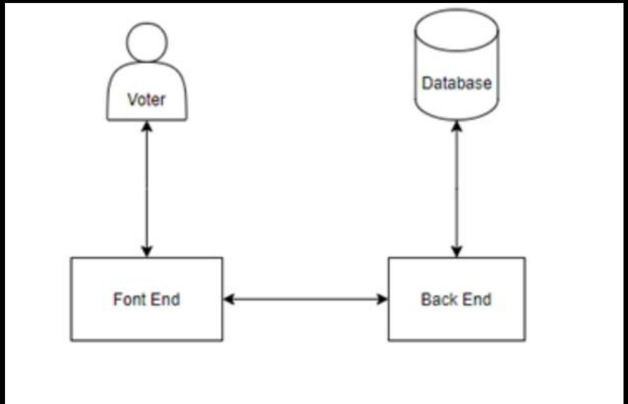
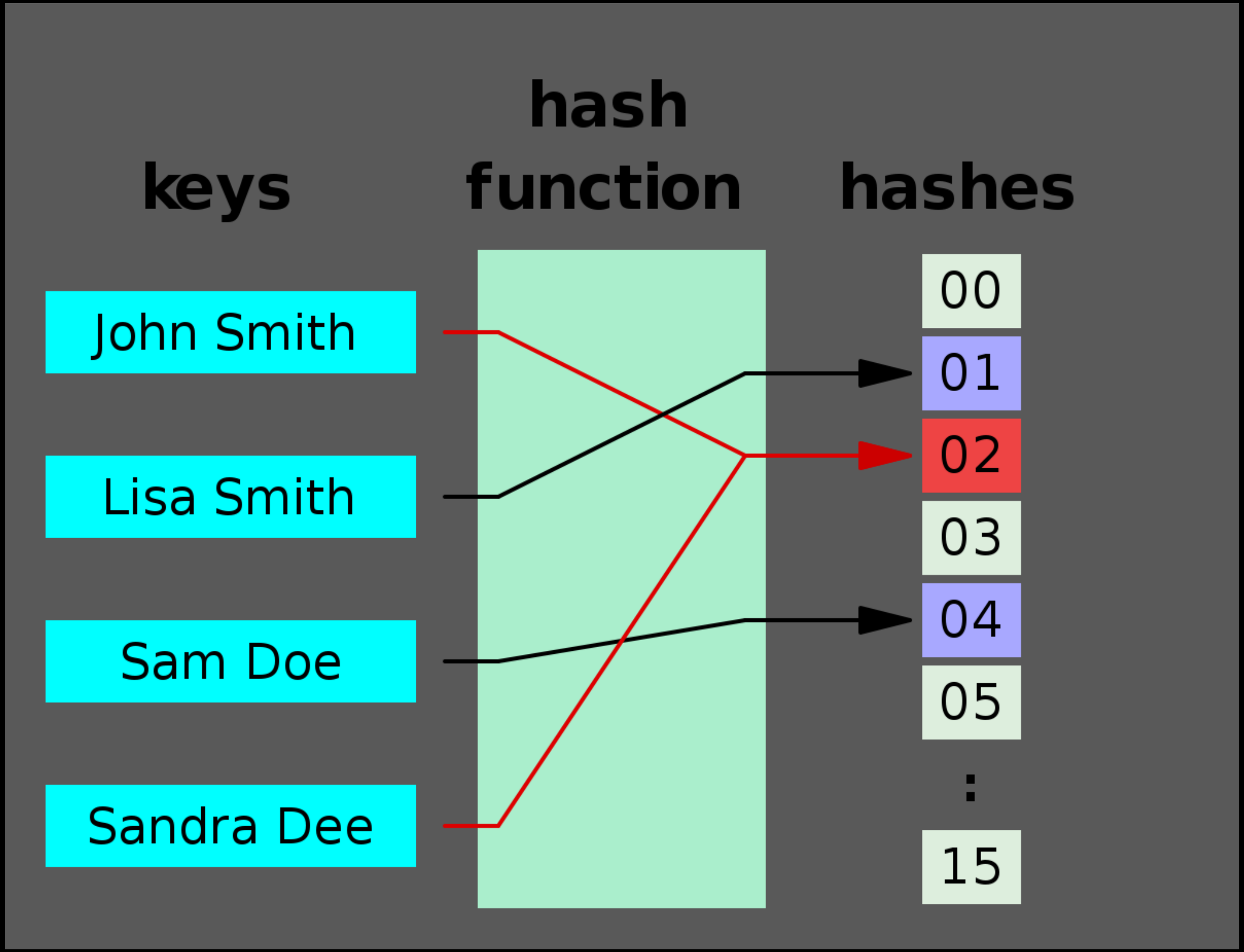


Figure 1.1: Traditional E-Voting Architecture

Blockchain

Hashes

- Same arbitrary length input
  -> same fixed-length output
- Hard to find input from output
- Hard to find two different inputs
  have same output
- efficient to compute the output



https://en.wikipedia.org/wiki/Hash_function

Merkle Tree

- Hash of hashes
- Leaf node:

  transaction in the blockchain
- Root Hash:

  summarize all the transactions

  in the blockchain.

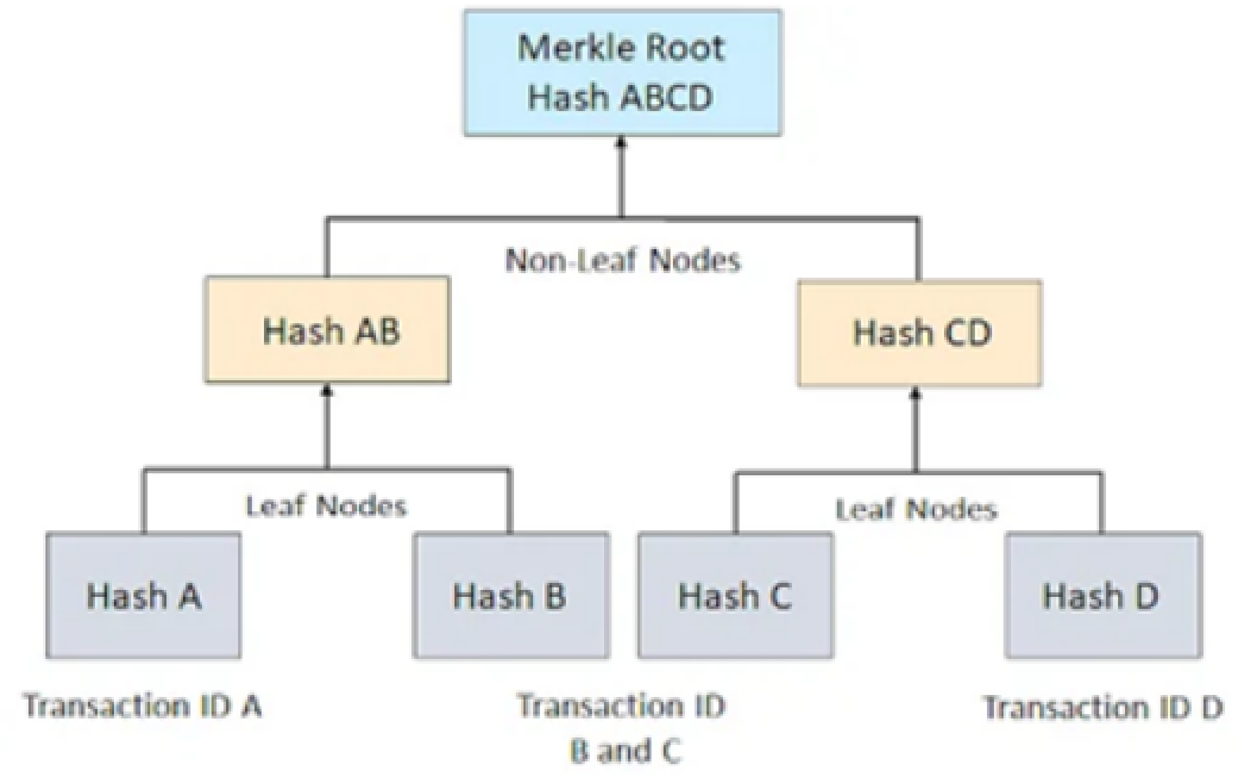
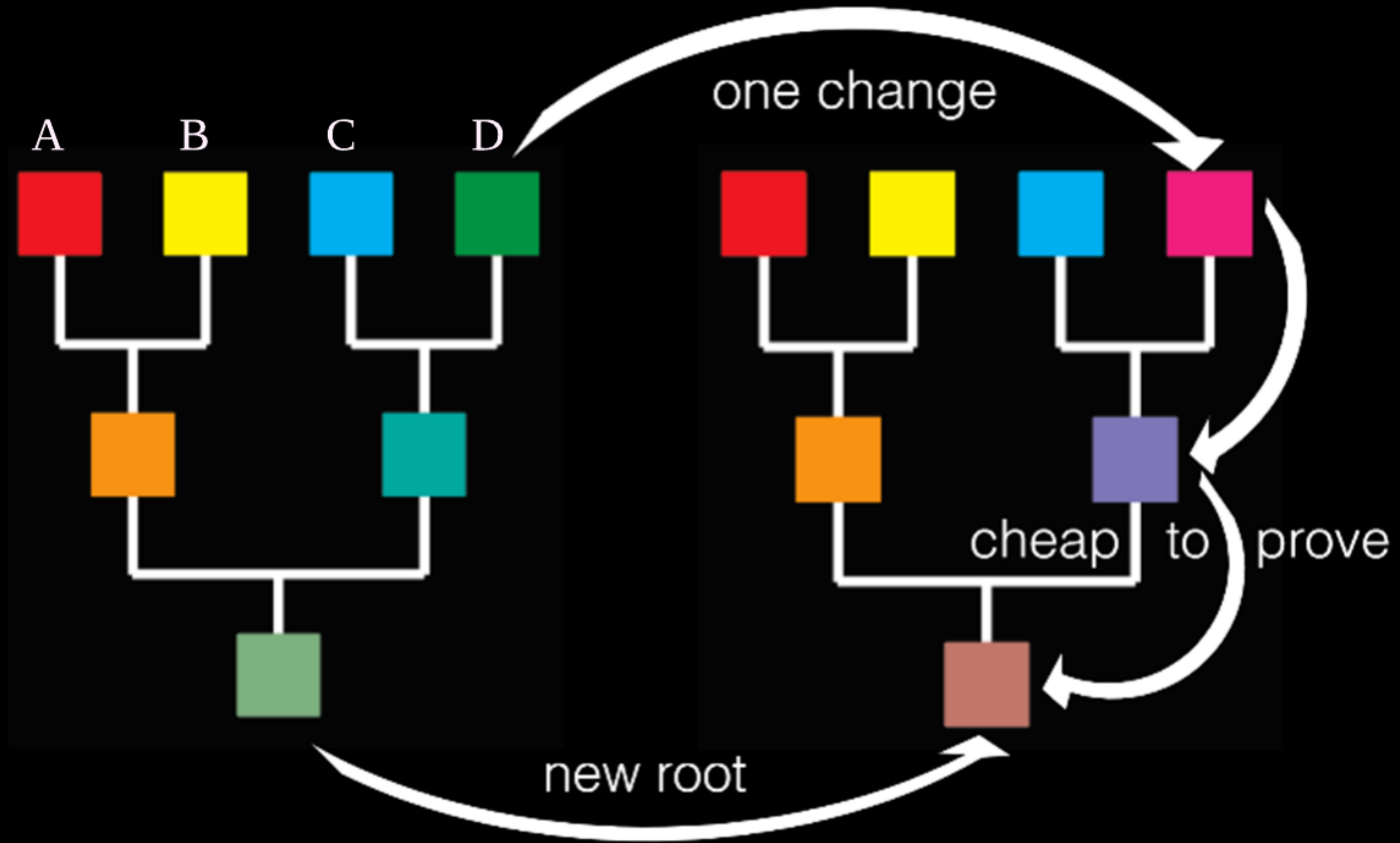


Figure 1.2: Merkle binary tree with 4 transactions

[Merkle Tree in Blockchain: What is it and How does it work | Simplilearn]

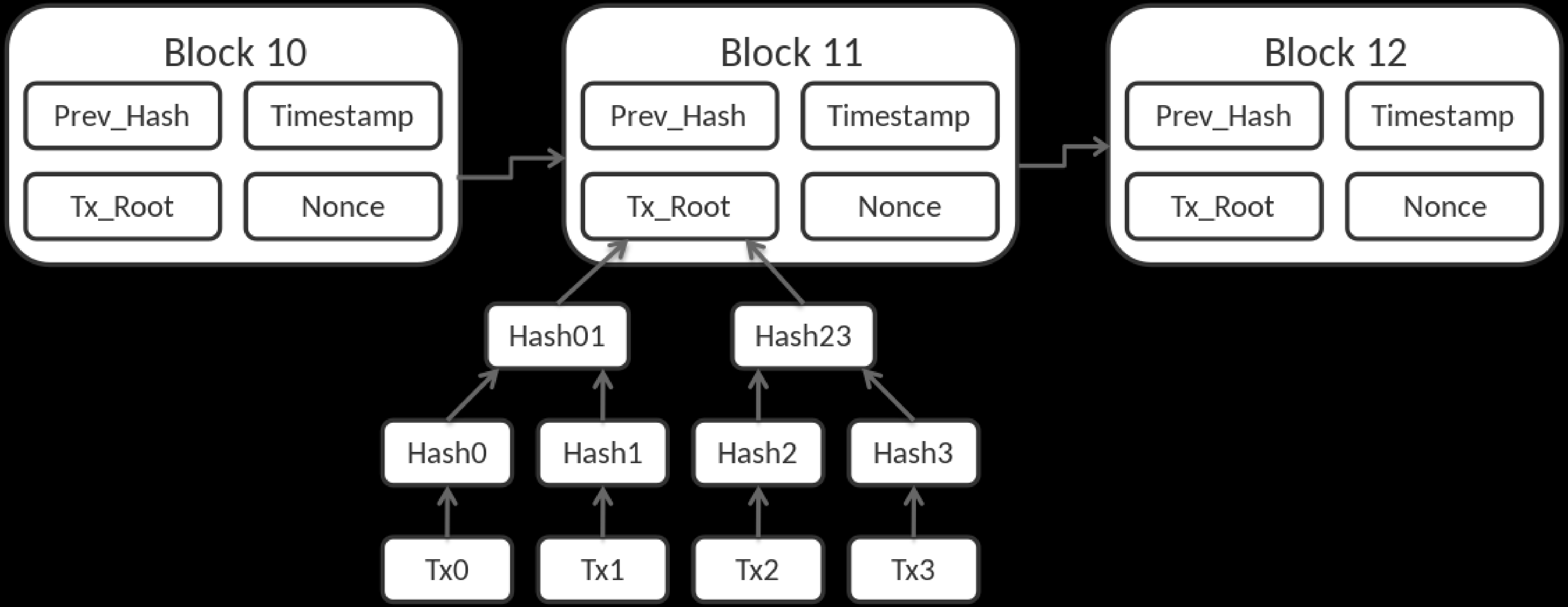
Merkle Tree



https://github.com/MetaMask/IPFS-Ethereum-Hackathon/blob/master/slides/01_DanFinlay_intro_to_ethereum_blockchains/DanFinlay-intro_to_ethereum.pdf

Blockchain

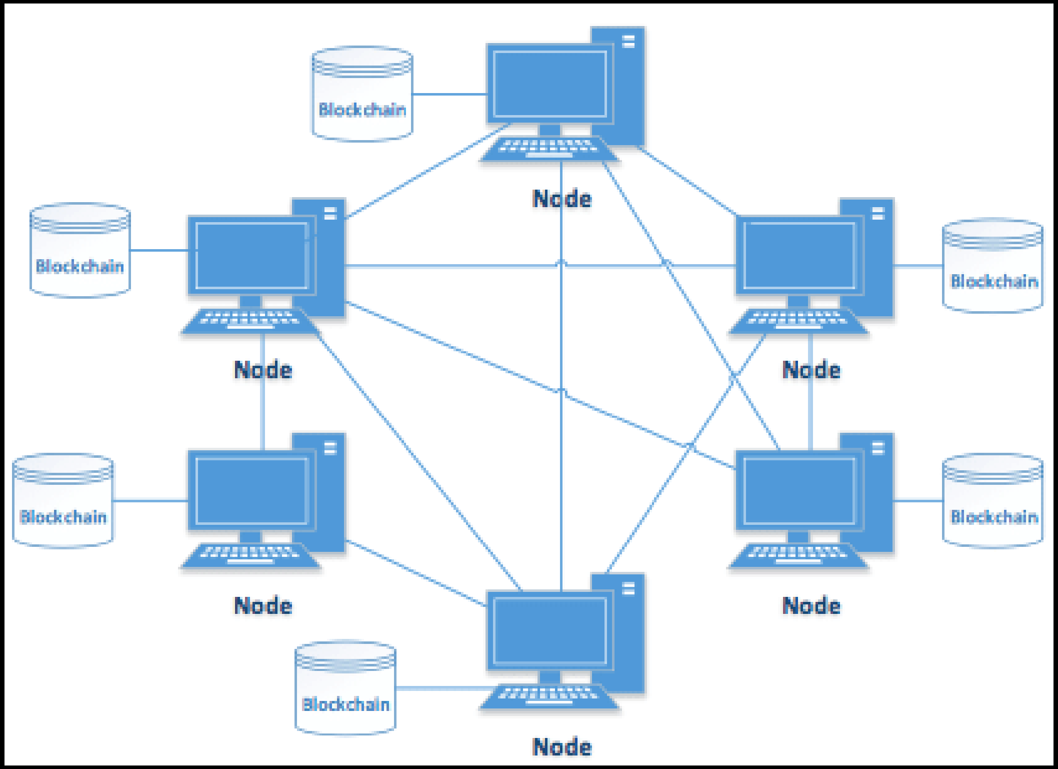
- Block of chain



https://en.wikipedia.org/wiki/Blockchain

Blockchain

- P2P network
  - Node: owns a copy of data in the network and can update the data in the network.
  - any update in the blockchain must be agreed upon by the consensus algorithm.

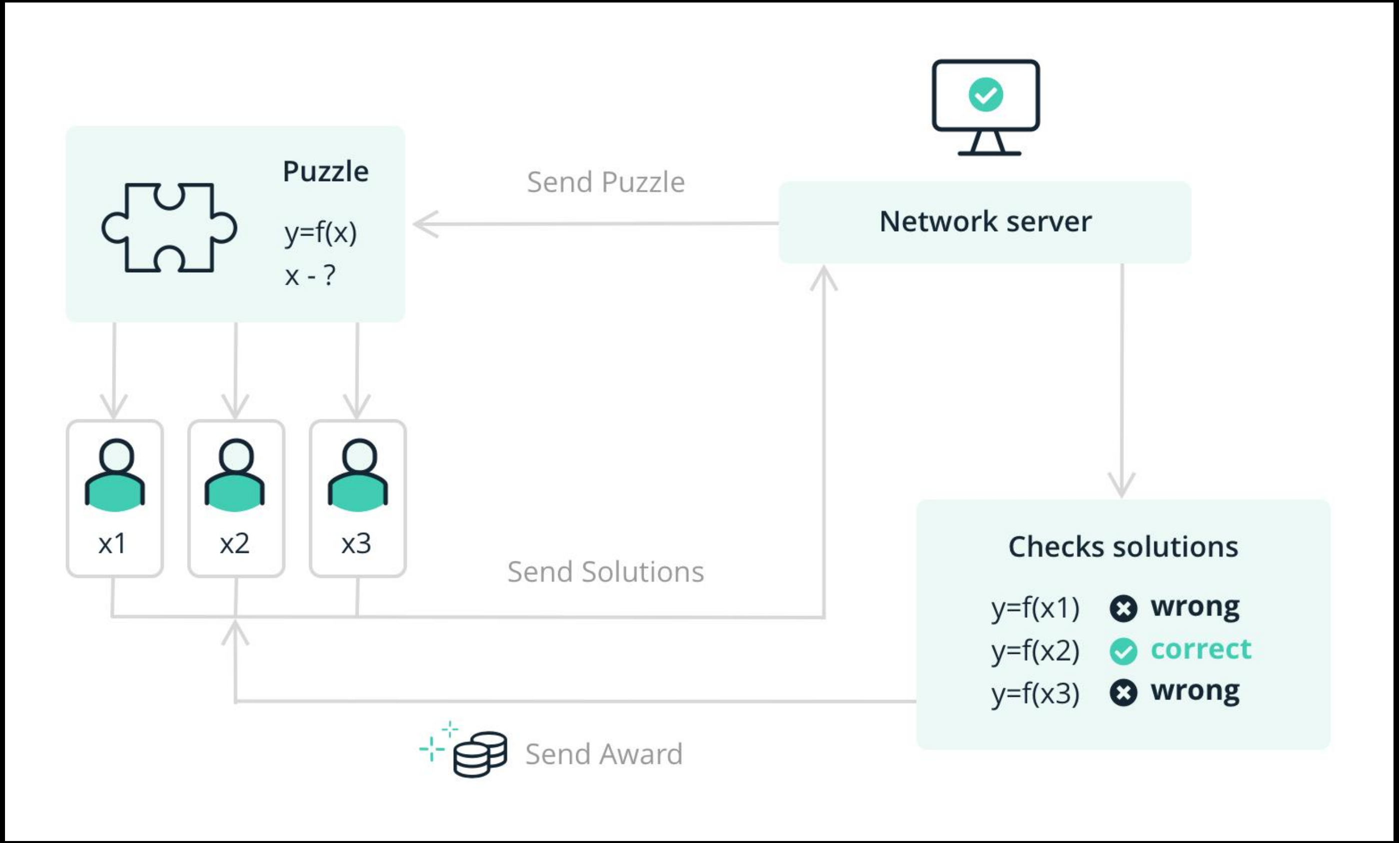


X : Dos Attacks

https://www.researchgate.net/figure/Blockchain-P2P-Network_fig1_320127088

Blockchain

- Consensus Algorithm (POW)
  - Puzzle:
    - Hard to solve
    - Easy to verify
  - *the majority decision is represented by the longest chain*



✓ : Immutability

https://www.ledger.com/academy/blockchain/what-is-proof-of-work

Blockchain

- Ethereum
  - Permissionless
  - Smart Contract
    - program stored on a blockchain
  - Accounts:
    - Externally Owned Account(EOA)
    - Contract Account(CA)

Cryptography Tools

Cryptography Tools

- Public Key Cryptosystem
  - Elliptic Curve Cryptography (ECC)
- Participant Registration
  - Schnorr’s Protocol
- Anonymous and Unique Voting
  - Linkable Ring Signature (LRS)
- Validating Message Sender
  - Elliptic Curve Digital Signature Algorithm (ECDSA)

Cryptography Tools

- Encryption / Decryption
  - ElGamal Encryption
  - Verifiable Decryption:
    - Chaum-Pedersen Protocol
- Key Distribution
  - Threshold Cryptosystem

Elliptic Curve Cryptography (ECC)

Elliptic Curve Cryptography (ECC)

- Elliptic Curve over Finite Field F_p
  - a plane algebraic curve that contains points {x, y}
  - Equation:
    - $y^2 = x^3 + ax + b\ (mod p\ )$
    - where p is a prime, $4a^3 + 27b^2 \neq 0$, $a, b, x, y \in F_p$ and an extra point O at "infinity".

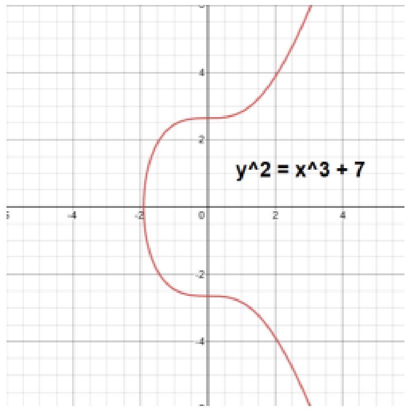


Figure 3.2: elliptic curves $E : y^2 = x^3 + 7$

Elliptic Curve Cryptography (ECC)

- Order $\boldsymbol{n}$
  - total number of points on $\boldsymbol{E(F_p)}$
- Subgroup $\boldsymbol{h}$
  - the points on the curve are divided into $\boldsymbol{h}$ number of subgroups
  - where the order of each subgroups is $\boldsymbol{r}$
- Generator / base point $\boldsymbol{G}$
  - a point on $\boldsymbol{E(F_p)}$ for generating other points on its subgroup by $\boldsymbol{rG}$

Elliptic Curve Cryptography (ECC)

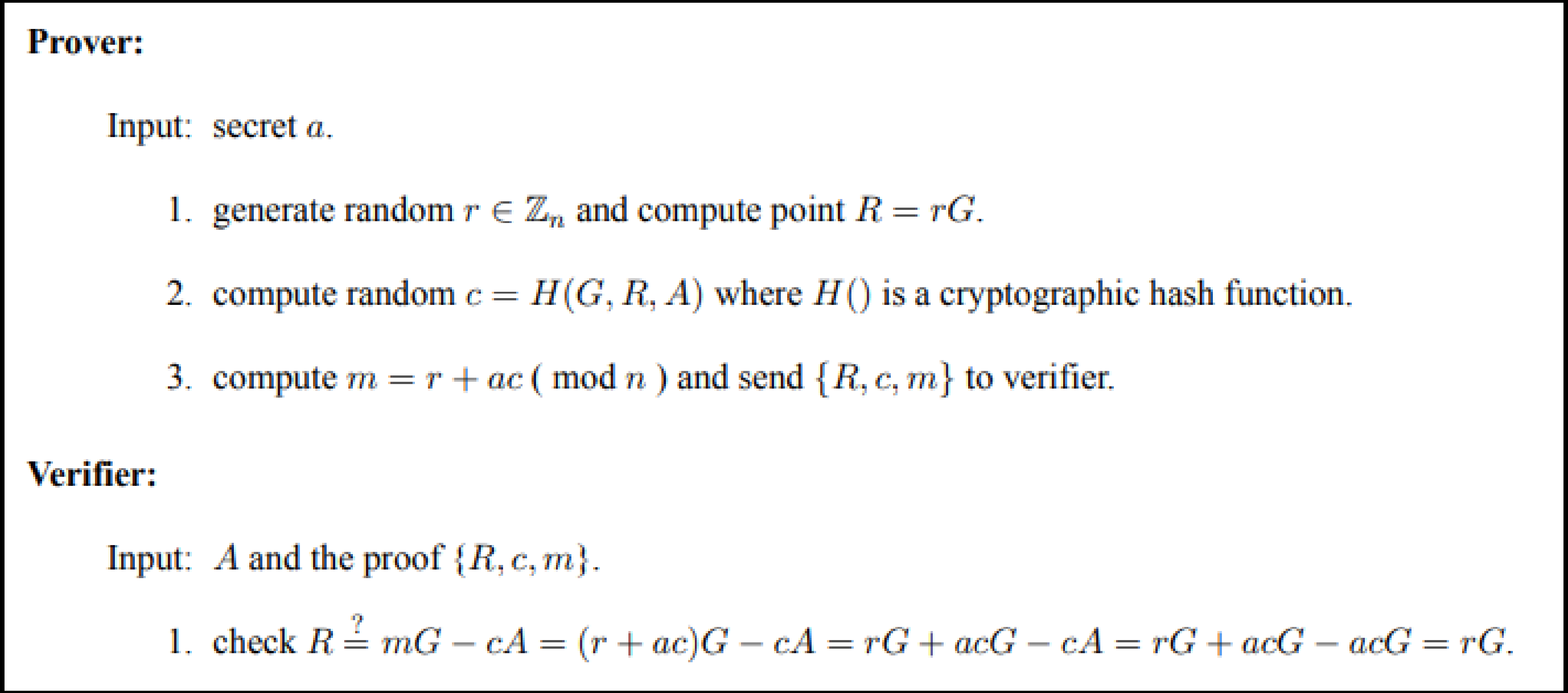
- Private Key
  - an integer $\boldsymbol{k}$
- Public Key
  - point $\mathrm{P} = \boldsymbol{kG}$
- Elliptic-Curve Discrete Logarithm Problem (ECDLP)
  - computational infeasible to find $\boldsymbol{k}$ that $\mathrm{P} = \boldsymbol{kG}$

Schnorr’s Protocol

Schnorr’s Protocol

- prover proves the knowledge of $\boldsymbol{a}$ where $\mathrm{A} = \boldsymbol{aG}$ is public without revealing $\boldsymbol{a}$.

Schnorr’s Protocol



Prover:

Input: secret a.

1. generate random $r \in \mathbb{Z}_n$ and compute point $R = rG$.
2. compute random $c = H(G, R, A)$ where $H()$ is a cryptographic hash function.
3. compute $m = r + ac \pmod{n}$ and send $\{R, c, m\}$ to verifier.

Verifier:

Input: A and the proof $\{R, c, m\}$.

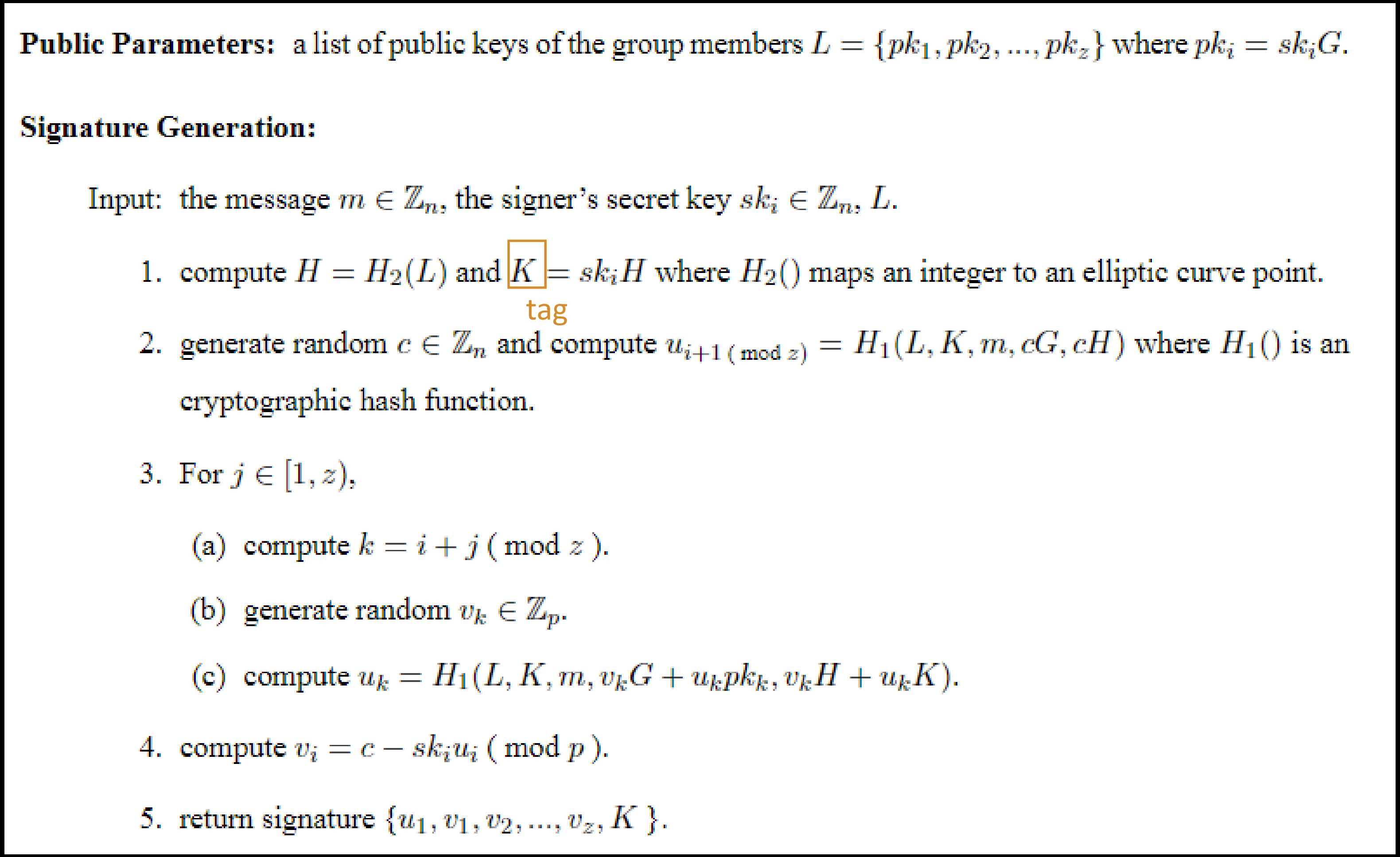
1. check $R \overset{?}{=} mG - cA = (r + ac)G - cA = rG + acG - cA = rG + acG - acG = rG$.

Linkable Ring Signature (LRS)

Linkable Ring Signature (LRS)

- Ring Signature
  - a group signature without a group manager and cooperation between group members that allows a signer to sign a message on behalf of the group without revealing which group member signed this message.
- LRS
  - a modification of a ring signature that detects whether the same signer generates two signatures.

LRS



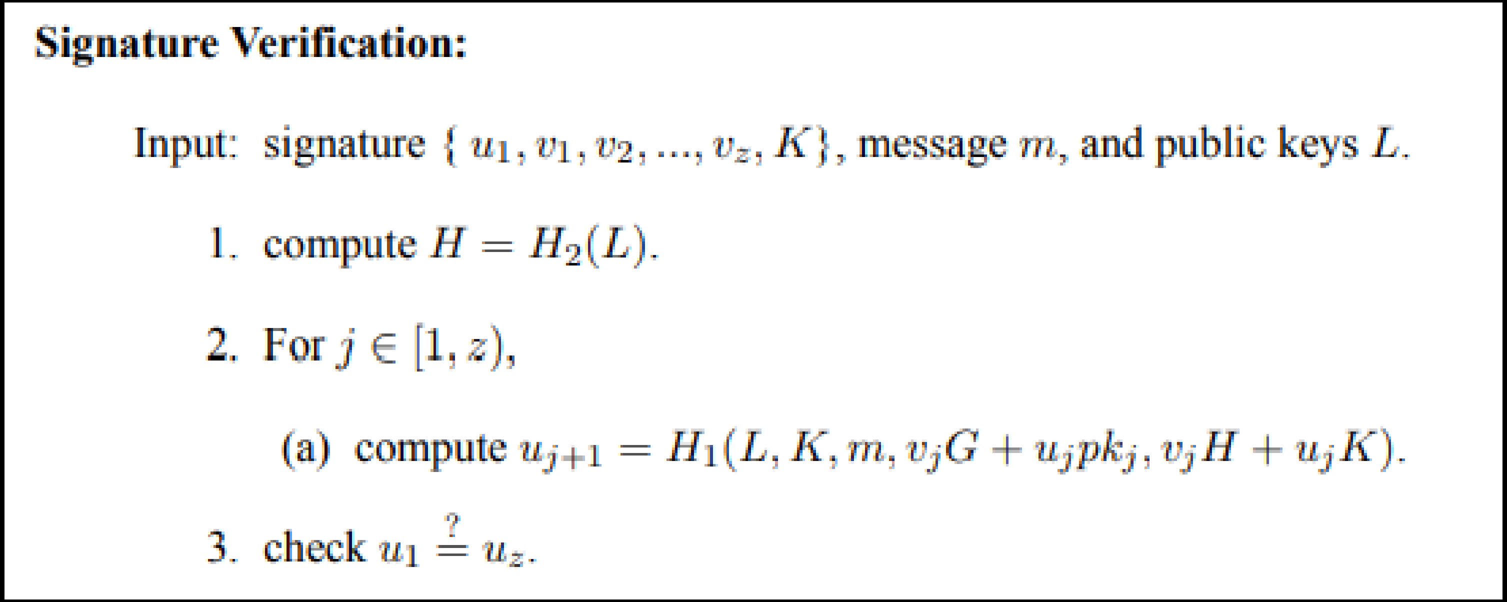
Public Parameters: a list of public keys of the group members $L = \{pk_1, pk_2, ..., pk_z\}$ where $pk_i = sk_iG$.

Signature Generation:

Input: the message $m \in \mathbb{Z}_n$, the signer's secret key $sk_i \in \mathbb{Z}_n$, L.

1. compute $H = H_2(L)$ and $\underset{\text{tag}}{\boxed{K}} = sk_iH$ where $H_2()$ maps an integer to an elliptic curve point.
2. generate random $c \in \mathbb{Z}_n$ and compute $u_{i+1 \pmod z} = H_1(L, K, m, cG, cH)$ where $H_1()$ is an cryptographic hash function.
3. For $j \in [1, z)$,
   (a) compute $k = i + j \pmod z$.
   (b) generate random $v_k \in \mathbb{Z}_p$.
   (c) compute $u_k = H_1(L, K, m, v_kG + u_kpk_k, v_kH + u_kK)$.
4. compute $v_i = c - sk_iu_i \pmod p$.
5. return signature $\{u_1, v_1, v_2, ..., v_z, K\}$.

LRS



Signature Verification:

Input: signature $\{ u_1, v_1, v_2, ..., v_z, K \}$, message m, and public keys L.

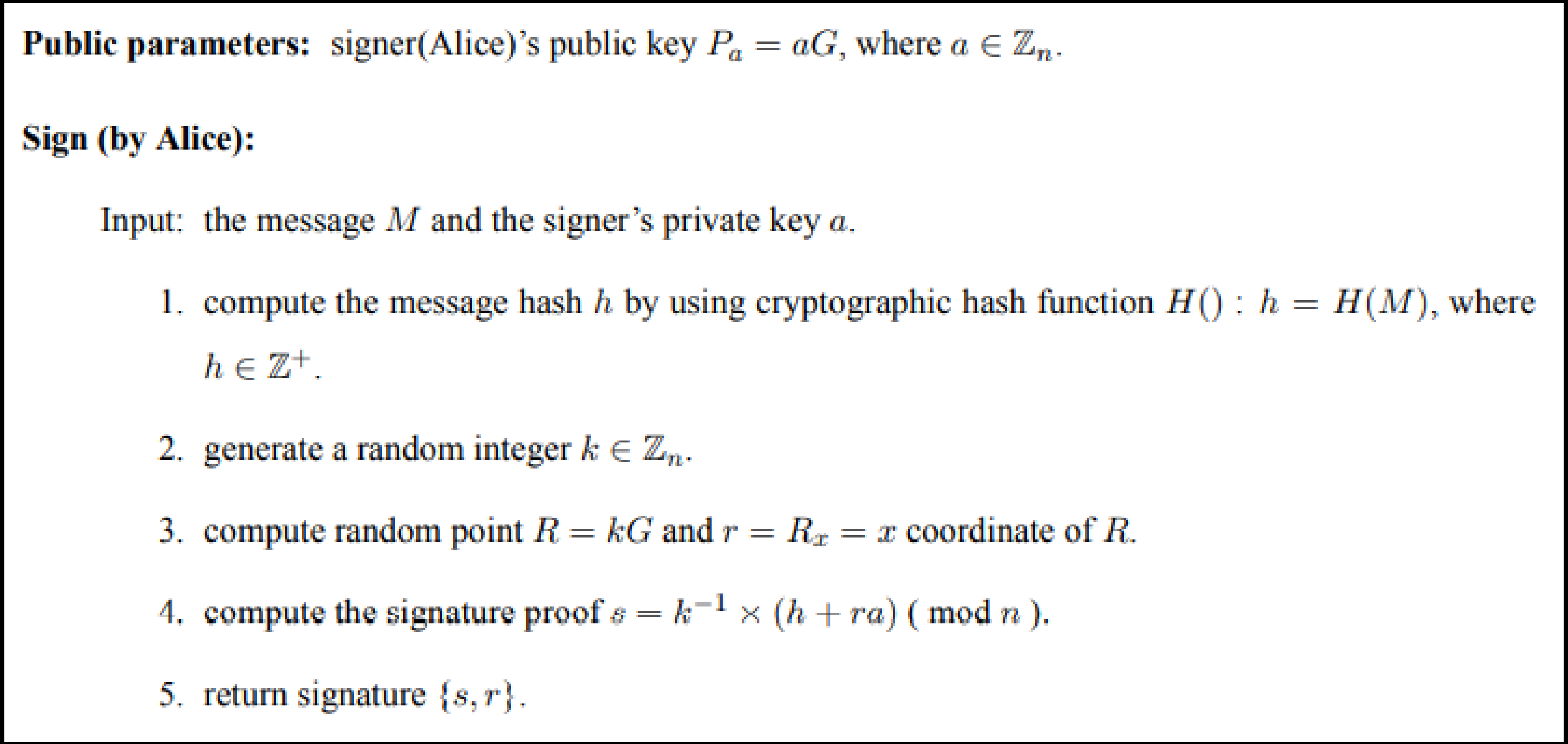
```
1. compute $H = H_2(L)$.
2. For $j \in [1, z)$,
    (a) compute $u_{j+1} = H_1(L, K, m, v_j G + u_j pk_j, v_j H + u_j K)$.
3. check $u_1 \overset{?}{=} u_z$.
```

Elliptic Curve Digital Signature Algorithm (ECDSA)

ECDSA

- offers the functionalities of a handwritten signature and data integrity
  - verifier can determine whether the message was modified
  - and the signer of the signed message

ECDSA



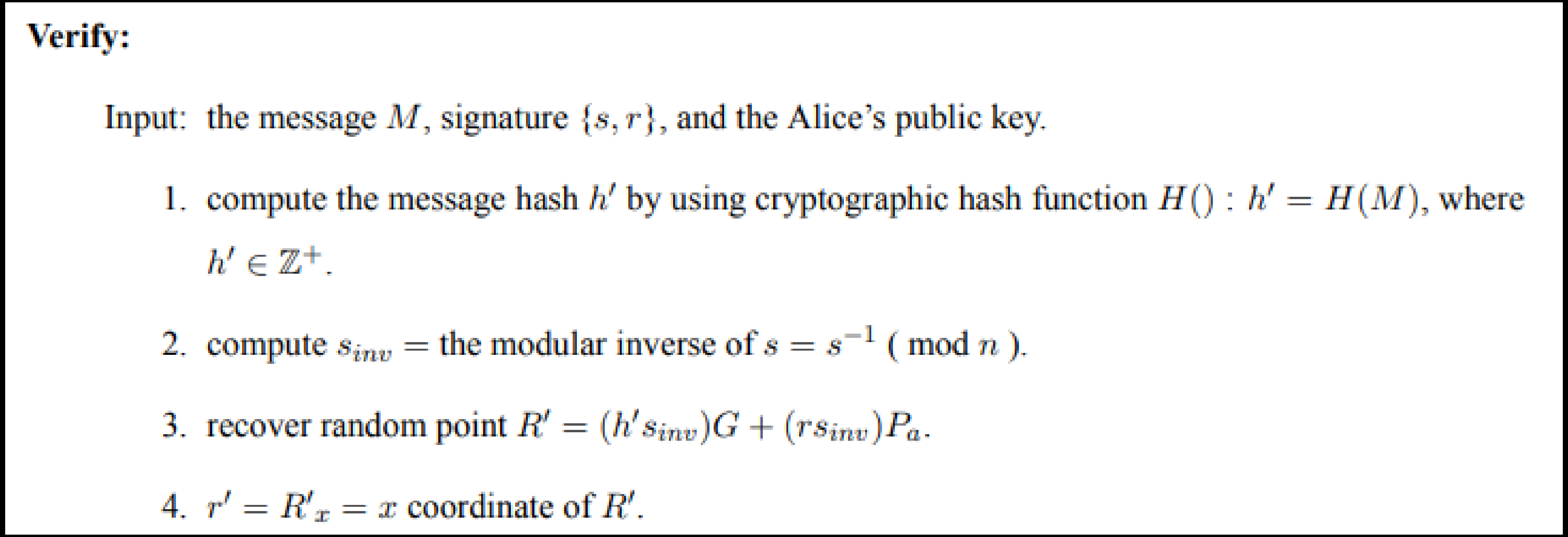
Public parameters: signer(Alice)'s public key $P_a = aG$, where $a \in \mathbb{Z}_n$.

Sign (by Alice):

Input: the message M and the signer's private key a.

1. compute the message hash h by using cryptographic hash function $H() : h = H(M)$, where $h \in \mathbb{Z}^+$.
2. generate a random integer $k \in \mathbb{Z}_n$.
3. compute random point $R = kG$ and $r = R_x = x$ coordinate of R.
4. compute the signature proof $s = k^{-1} \times (h + ra) \pmod{n}$.
5. return signature $\{s, r\}$.

ECDSA



Verify:

Input: the message M, signature $\{s, r\}$, and the Alice's public key.

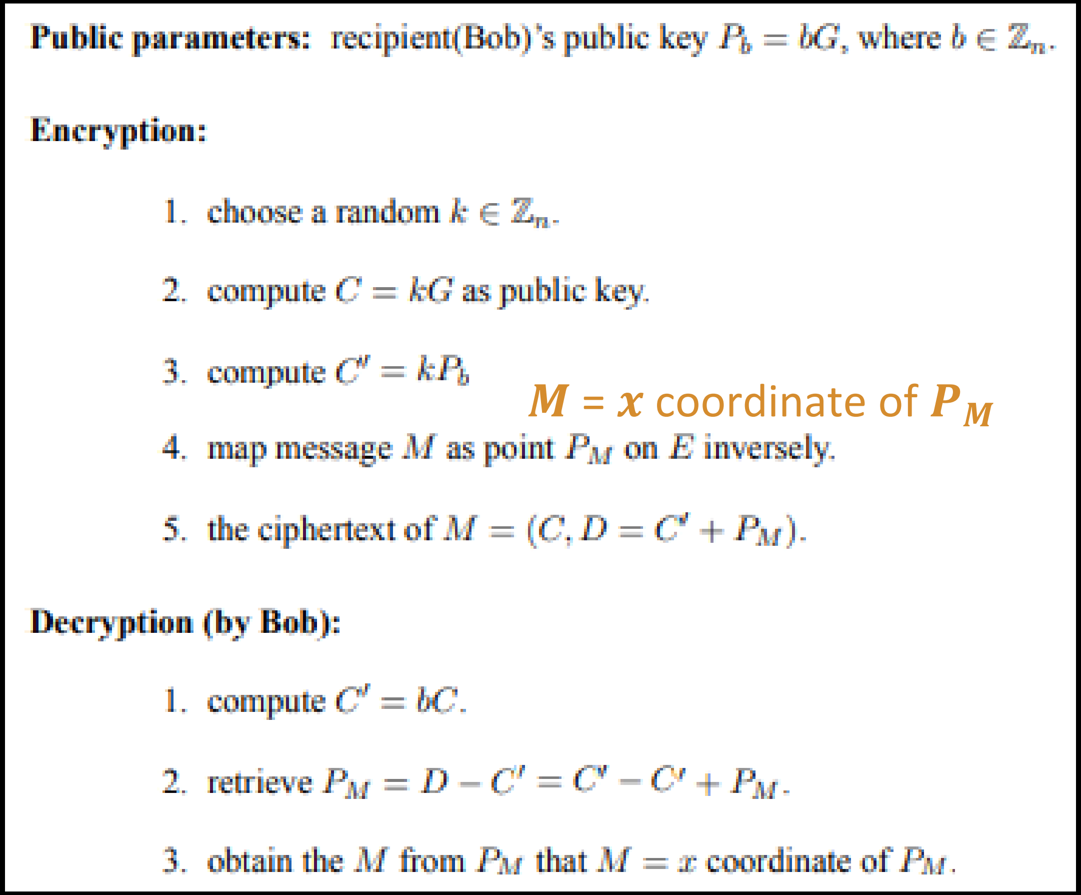
1. compute the message hash h' by using cryptographic hash function $H() : h' = H(M)$, where $h' \in \mathbb{Z}^+$.
2. compute s_{inv} = the modular inverse of $s = s^{-1} \pmod{n}$.
3. recover random point $R' = (h' s_{inv})G + (r s_{inv})P_a$.
4. $r' = R'_x = x$ coordinate of R'.

ElGamal Encryption

ElGamal Encryption

- asymmetric encryption scheme that encrypts a message by a one-time-key

ElGamal Encryption



Public parameters: recipient(Bob)'s public key $P_b = bG$, where $b \in \mathbb{Z}_n$.

Encryption:

1. choose a random $k \in \mathbb{Z}_n$.
2. compute $C = kG$ as public key.
3. compute $C' = kP_b$
4. map message M as point P_M on E inversely.
5. the ciphertext of $M = (C, D = C' + P_M)$.

M = x coordinate of P_M

Decryption (by Bob):

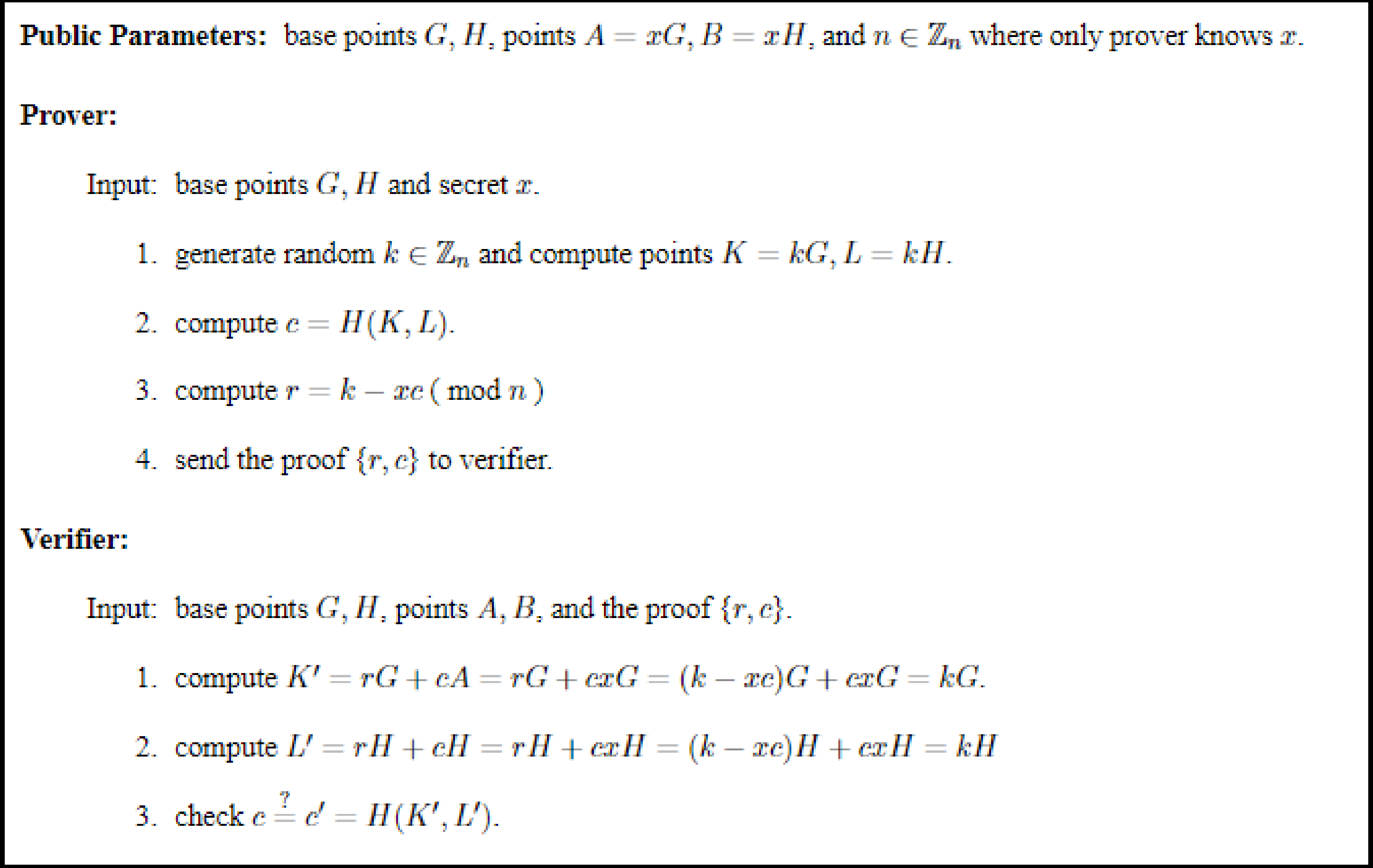
1. compute $C' = bC$.
2. retrieve $P_M = D - C' = C' - C' + P_M$.
3. obtain the M from P_M that $M = x$ coordinate of P_M.

Chaum-Pedersen Protocol

Chaum-Pedersen Protocol

- proof of knowledge for the equality of discrete logarithms
  - $\boldsymbol{log_G(xG) = log_H(xH)}$
  - where $\boldsymbol{G}, \boldsymbol{H}$ are two different generators on curve $\boldsymbol{E}$

Chaum-Pedersen Protocol



Public Parameters: base points G, H, points $A = xG, B = xH$, and $n \in \mathbb{Z}_n$ where only prover knows x.

Prover:

Input: base points G, H and secret x.

1. generate random $k \in \mathbb{Z}_n$ and compute points $K = kG, L = kH$.
2. compute $c = H(K, L)$.
3. compute $r = k - xc \pmod{n}$
4. send the proof $\{r, c\}$ to verifier.

Verifier:

Input: base points G, H, points A, B, and the proof $\{r, c\}$.

1. compute $K' = rG + cA = rG + cxG = (k - xc)G + cxG = kG$.
2. compute $L' = rH + cH = rH + cxH = (k - xc)H + cxH = kH$
3. check $c \overset{?}{=} c' = H(K', L')$.

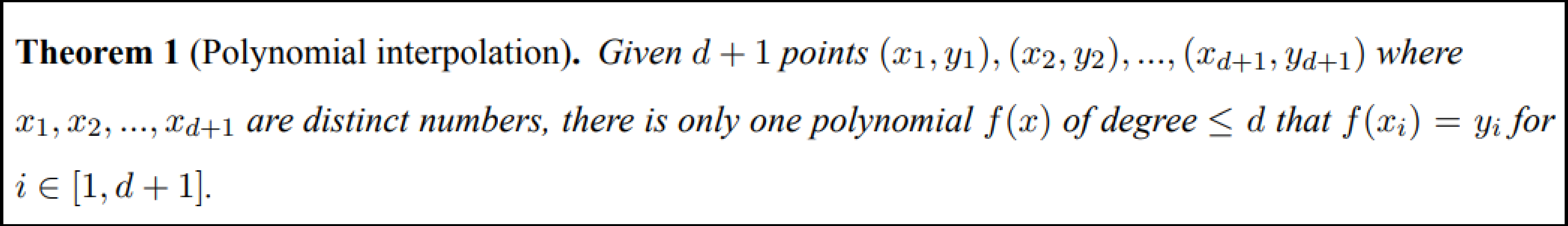
Threshold Cryptosystem

Polynomial

- A polynomial $\boldsymbol{f(x)}$ is a mathematical expression in the form expression in the form:
  - $\boldsymbol{a_n x^n + a_{n-1} x^{n-1} + a_{n-2} x^{n-2} + \cdots + a0}$
- Coefficients
  - $\boldsymbol{a_n, a_{n-1}, a_{n-2}, \ldots, a0}$
- Degree
  - the highest exponent of $\boldsymbol{x}$

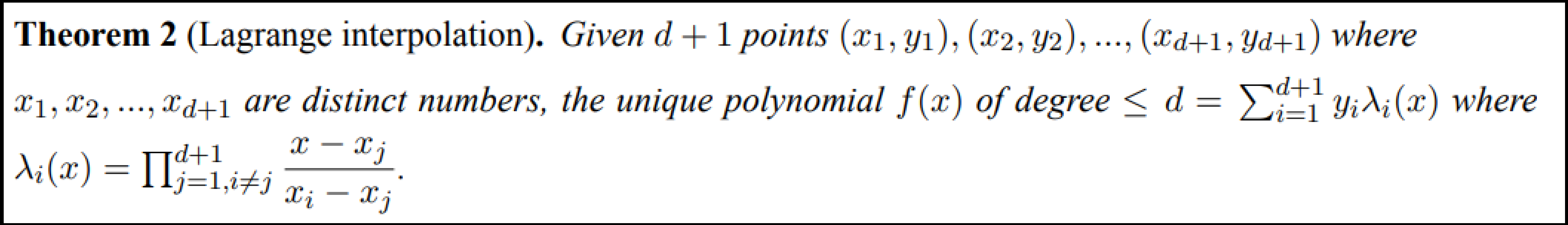
Polynomial

- Polynomial Interpolation



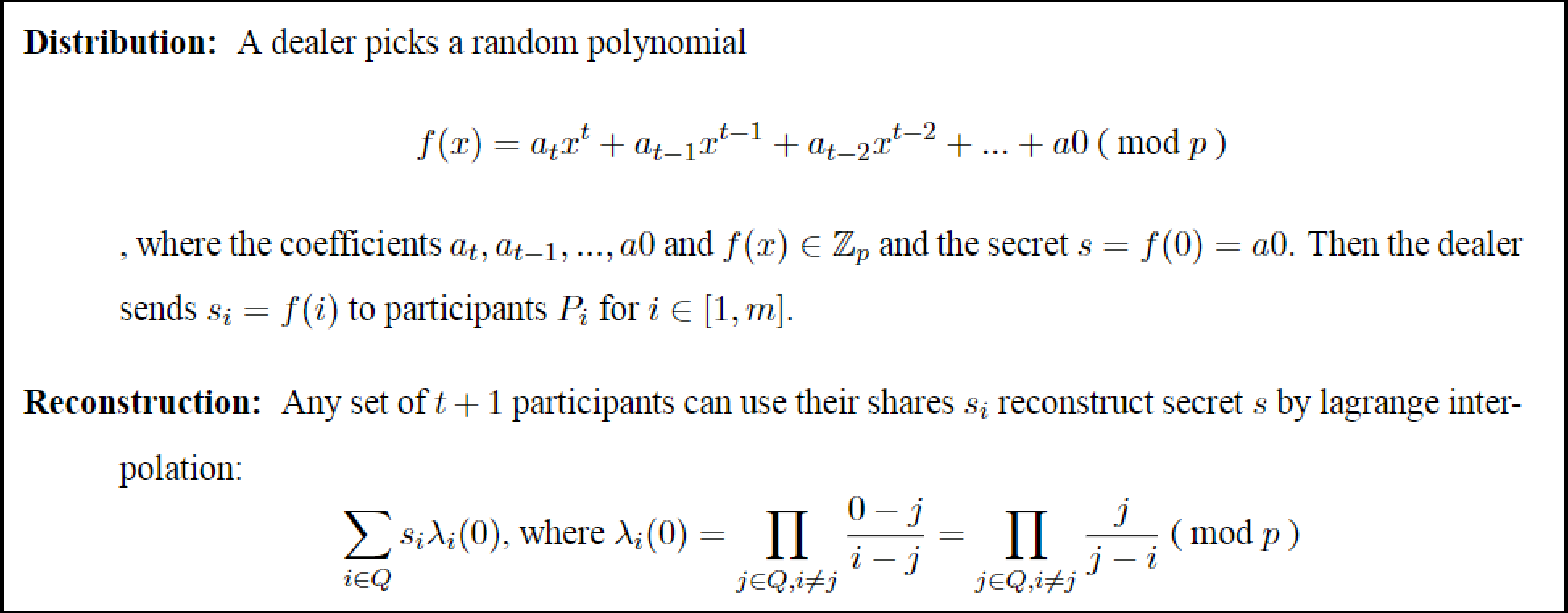
Theorem 1 (Polynomial interpolation). *Given* $d+1$ *points* $(x_1, y_1), (x_2, y_2), ..., (x_{d+1}, y_{d+1})$ *where* $x_1, x_2, ..., x_{d+1}$ *are distinct numbers, there is only one polynomial* $f(x)$ *of degree* $\leq d$ *that* $f(x_i) = y_i$ *for* $i \in [1, d+1]$.

- Lagrange interpolation



Theorem 2 (Lagrange interpolation). *Given* $d+1$ *points* $(x_1, y_1), (x_2, y_2), ..., (x_{d+1}, y_{d+1})$ *where* $x_1, x_2, ..., x_{d+1}$ *are distinct numbers, the unique polynomial* $f(x)$ *of degree* $\leq d = \sum_{i=1}^{d+1} y_i \lambda_i(x)$ *where* $\lambda_i(x) = \prod_{j=1, i \neq j}^{d+1} \dfrac{x - x_j}{x_i - x_j}$.

Shamir Threshold Scheme



Distribution: A dealer picks a random polynomial

$$f(x) = a_t x^t + a_{t-1} x^{t-1} + a_{t-2} x^{t-2} + ... + a0 \pmod{p}$$

, where the coefficients $a_t, a_{t-1}, ..., a0$ and $f(x) \in \mathbb{Z}_p$ and the secret $s = f(0) = a0$. Then the dealer sends $s_i = f(i)$ to participants P_i for $i \in [1, m]$.

Reconstruction: Any set of $t + 1$ participants can use their shares s_i reconstruct secret s by lagrange interpolation:

$$\sum_{i \in Q} s_i \lambda_i(0), \text{ where } \lambda_i(0) = \prod_{j \in Q, i \neq j} \frac{0 - j}{i - j} = \prod_{j \in Q, i \neq j} \frac{j}{j - i} \pmod{p}$$

Shamir Threshold Scheme

- Even t participant pool their shares together, they still cannot know the secret

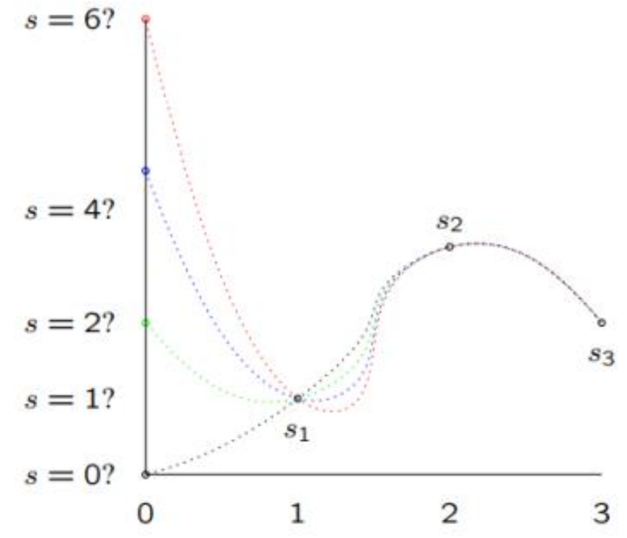


Figure 3.7: Security of Shamir's scheme illustrated: 5 degree-3 polynomials that can interpolates s_1, s_2, s_3.

www.tcs.hut.fi/Studies/T-79.159/2004/slides/L9.pdf

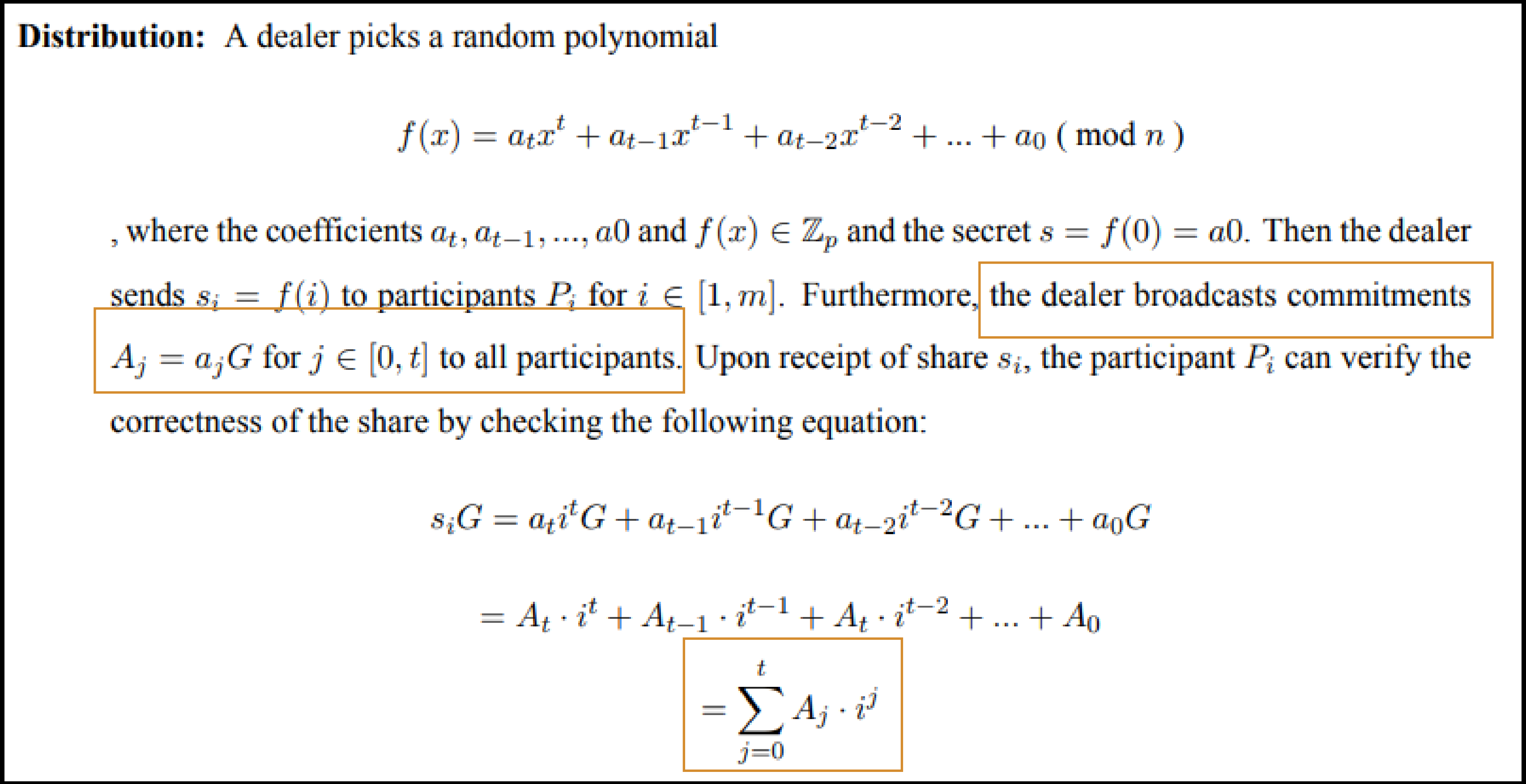
Shamir Threshold Scheme

- Problem
  - A dealer knows the secret
  - A dealer sends incorrect shares to some or all participants

Feldman VSS

- an extension of Shamir’s secret sharing scheme
- the dealer not only sends the share $\boldsymbol{s_i}$ to participant $\boldsymbol{P_i}$ but also broadcasts a verification value to all participants such that the participants can use them to validate their shares

Feldman VSS



Distribution: A dealer picks a random polynomial

$$f(x) = a_t x^t + a_{t-1} x^{t-1} + a_{t-2} x^{t-2} + \dots + a_0 \pmod{n}$$

, where the coefficients $a_t, a_{t-1}, \dots, a0$ and $f(x) \in \mathbb{Z}_p$ and the secret $s = f(0) = a0$. Then the dealer sends $s_i = f(i)$ to participants P_i for $i \in [1, m]$. Furthermore, the dealer broadcasts commitments $A_j = a_j G$ for $j \in [0, t]$ to all participants. Upon receipt of share s_i, the participant P_i can verify the correctness of the share by checking the following equation:

$$s_i G = a_t i^t G + a_{t-1} i^{t-1} G + a_{t-2} i^{t-2} G + \dots + a_0 G$$

$$= A_t \cdot i^t + A_{t-1} \cdot i^{t-1} + A_t \cdot i^{t-2} + \dots + A_0$$

$$\boxed{= \sum_{j=0}^{t} A_j \cdot i^j}$$

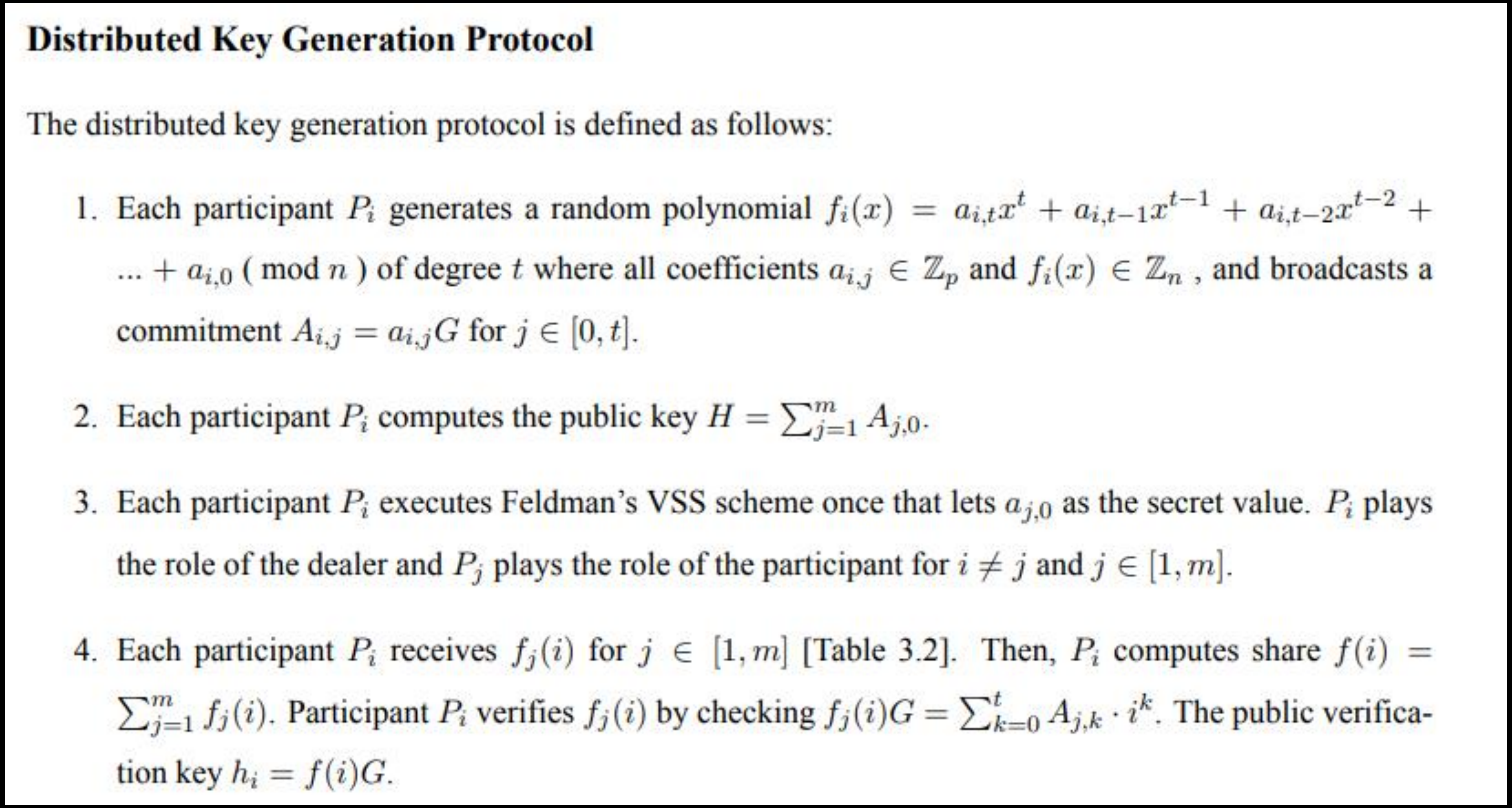
Feldman VSS

- Problem
  - A dealer knows the secret

Threshold Cryptosystem

- No dealer
    - Each participant plays the role of the dealer

Threshold Cryptosystem

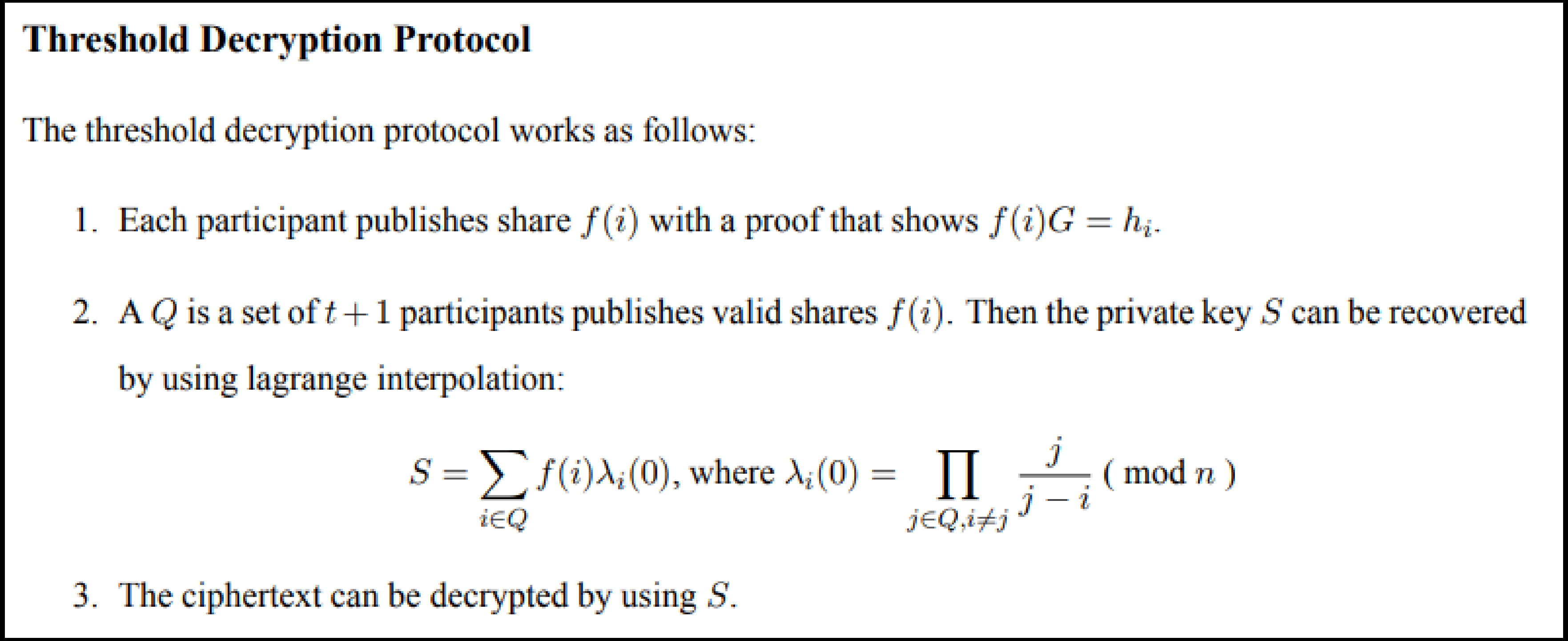


Distributed Key Generation Protocol

The distributed key generation protocol is defined as follows:

1. Each participant P_i generates a random polynomial $f_i(x) = a_{i,t}x^t + a_{i,t-1}x^{t-1} + a_{i,t-2}x^{t-2} + ... + a_{i,0}$ (mod n) of degree t where all coefficients $a_{i,j} \in \mathbb{Z}_p$ and $f_i(x) \in \mathbb{Z}_n$, and broadcasts a commitment $A_{i,j} = a_{i,j}G$ for $j \in [0, t]$.
2. Each participant P_i computes the public key $H = \sum_{j=1}^{m} A_{j,0}$.
3. Each participant P_i executes Feldman's VSS scheme once that lets $a_{j,0}$ as the secret value. P_i plays the role of the dealer and P_j plays the role of the participant for $i \neq j$ and $j \in [1, m]$.
4. Each participant P_i receives $f_j(i)$ for $j \in [1, m]$ [Table 3.2]. Then, P_i computes share $f(i) = \sum_{j=1}^{m} f_j(i)$. Participant P_i verifies $f_j(i)$ by checking $f_j(i)G = \sum_{k=0}^{t} A_{j,k} \cdot i^k$. The public verification key $h_i = f(i)G$.

Threshold Cryptosystem



Threshold Decryption Protocol

The threshold decryption protocol works as follows:

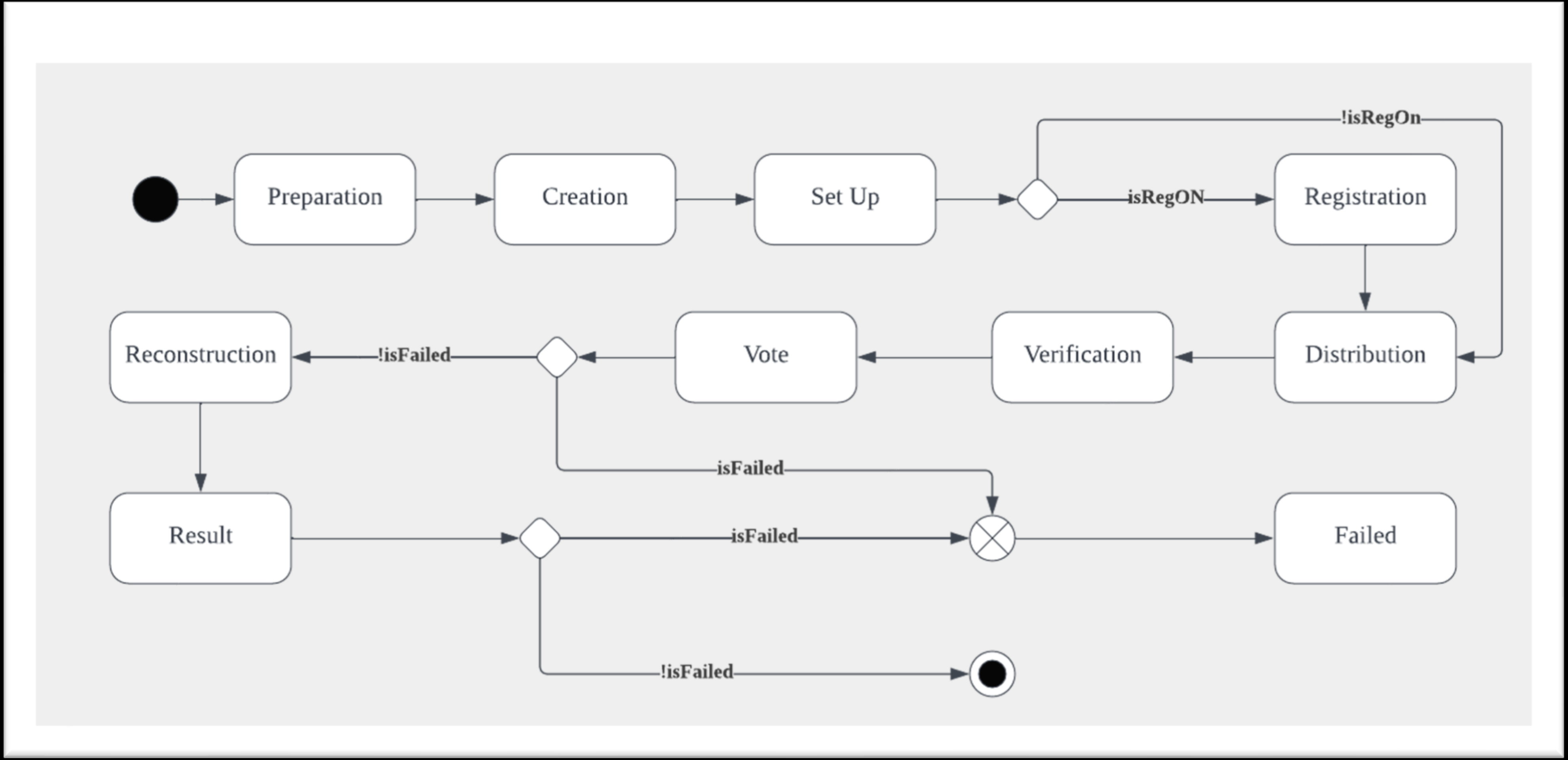
1. Each participant publishes share $f(i)$ with a proof that shows $f(i)G = h_i$.
2. A Q is a set of $t+1$ participants publishes valid shares $f(i)$. Then the private key S can be recovered by using lagrange interpolation:

$$S = \sum_{i \in Q} f(i)\lambda_i(0), \text{ where } \lambda_i(0) = \prod_{j \in Q, i \neq j} \frac{j}{j-i} \pmod{n}$$

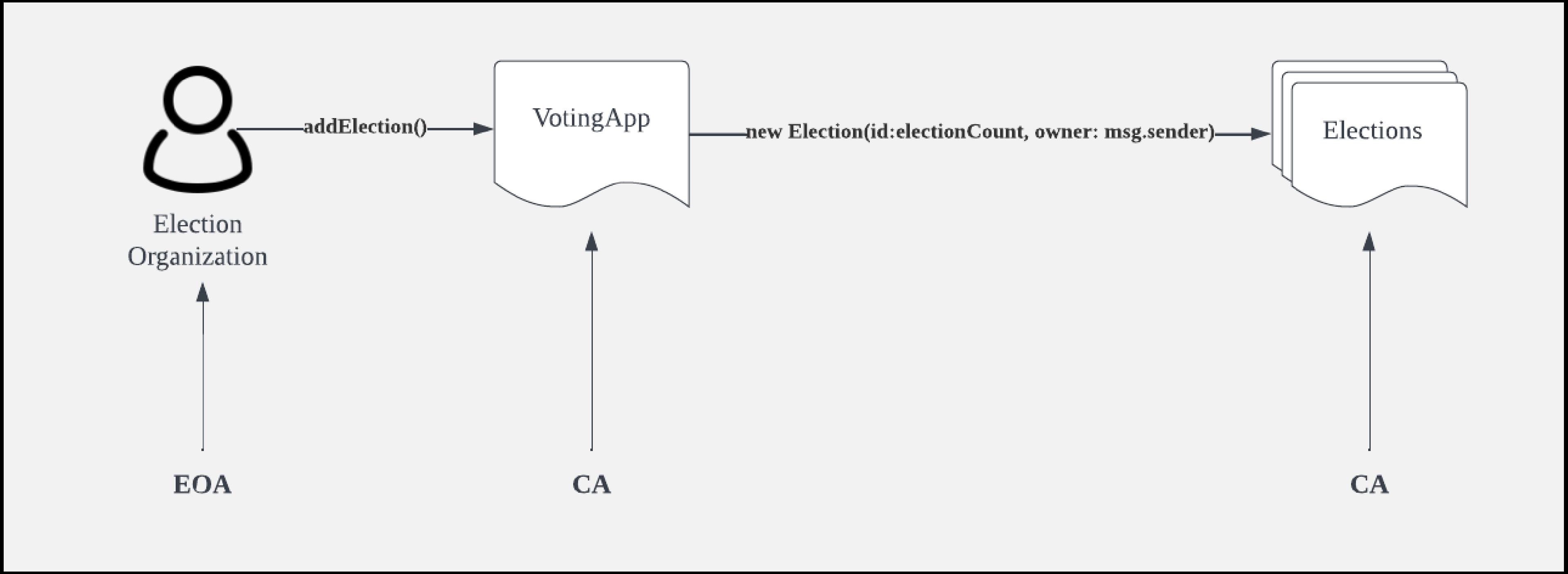
3. The ciphertext can be decrypted by using S.

Blockchain based E-Voting

An Overview of E-Voting Stages



Creation Stage



Setup

- Candidates $\boldsymbol{C_i}$ for $\boldsymbol{i} \in [\boldsymbol{1}, \boldsymbol{d}]$
- Participants' public keys $\boldsymbol{pk_i}$ for $\boldsymbol{i} \in [\boldsymbol{1}, \boldsymbol{m}]$
- minShares $\boldsymbol{t}$
- isRegOn
- regInfo
  - $\boldsymbol{R_i} = \boldsymbol{r_i G}$ for $\boldsymbol{i} \in [\boldsymbol{1}, \boldsymbol{h}]$ where $\boldsymbol{r_i}$ is register i's personal data
- Timers
- …

Registration Stage

- Register
  - Generate a key pair $(\boldsymbol{sk_{m+1}}, \boldsymbol{pk_{m+1}})$
  - Find his $\boldsymbol{R_i}$
  - generate a schnorr proof $\mathrm{p} = \mathrm{schnorrProve}(\boldsymbol{r_i})$
  - Send $(\mathrm{pk_{m+1}}, \mathrm{i}, \mathrm{p})$

- Election
  - Verify schnorr proof $\mathrm{schnorrVer}(\mathrm{R_i}, \mathrm{p})$
  - Add $\mathrm{pk_{m+1}}$
  - Update $\boldsymbol{m} = \boldsymbol{m} + 1$

Distribution Stage

- Each participant
  - generates a random polynomial $\boldsymbol{f_i(x)}$ of degree $\boldsymbol{t\ -1}$ where coefficient is $\boldsymbol{a_{i,j}}$ for $\boldsymbol{j} \in [\mathbf{1}, \boldsymbol{t} - \mathbf{1}]$ and compute commitment $\boldsymbol{A_{i,j} = a_{i,j} \cdot G}$
  - $\mathrm{sigA_{i,j}}$: ECDSA signature of $\boldsymbol{A_{i,j}}$ for $\boldsymbol{j} \in [\mathbf{1}, \boldsymbol{t} - \mathbf{1}]$
  - $\boldsymbol{F_i(k)}$: ElGamal ciphertext of $\boldsymbol{f_i(k)}$
    - **$\boldsymbol{F_i(k)}$ can only be decrypted by participant $\boldsymbol{P_k}$ using his secret key $\boldsymbol{sk_k}$**
  - $\boldsymbol{sigF_i(k)}$: ECDSA signature of $\boldsymbol{F_i(k)}$ for $\boldsymbol{k} \in [\mathbf{1}, \boldsymbol{m}]$
  - Send each $\boldsymbol{A_{i,j}}, \mathrm{sigA_{i,j}}, \boldsymbol{F_i(k)}, \boldsymbol{sigF_i(k)}$

Distribution Stage

- Election
  - Verify signature $\mathrm{sigA}_{\mathrm{i,j}}$ and store $\boldsymbol{A_{i,j}}$
  - Verify signature $\boldsymbol{sigF_i(k)}$ and store $\boldsymbol{F_i(k)}$

Verification Stage

- Report Non-contributed Participant
- Report Malicious Participant

Report Non-contributed Participant

- Election
  - check each participant $\boldsymbol{P_i}$ whether he sends his commitment $\boldsymbol{A_{i,j}}$ for $\boldsymbol{j} \in [\mathbf{1}, \boldsymbol{t} - \mathbf{1}]$
  - check each participant $\boldsymbol{P_i}$ whether he sends $\boldsymbol{F_i(k)}$ to participant $\boldsymbol{P_k}$ for $\boldsymbol{k} \in [\mathbf{1}, \boldsymbol{m}]$
  - Put them in disqualified participants set $\{\boldsymbol{P_i}\} \cup \boldsymbol{Q}$

Report Malicious Participant

- Each Participant $\boldsymbol{P_i}$
  - Decrypt $\boldsymbol{F_j(i)} = \left(\boldsymbol{C}, \boldsymbol{D} = \boldsymbol{C'} + \boldsymbol{f_j(i)}\right)$ and verify $\boldsymbol{f_j(i)}$ by checking $\boldsymbol{f_j(i)G} = \sum_{\boldsymbol{k=0}}^{\boldsymbol{t-1}} \boldsymbol{A_{j,k}} \cdot \boldsymbol{i^k}$ for $\boldsymbol{j} \in [\boldsymbol{1}, \boldsymbol{m}]$ and $\boldsymbol{P_j} \notin \boldsymbol{Q}$
  - compute proof of correct decryption key $\boldsymbol{p} = \boldsymbol{cpProve}\left(\boldsymbol{G}, \boldsymbol{F_j(i)}.\boldsymbol{C}, \boldsymbol{sk_i}\right)$ and the decryption key is $\boldsymbol{C'} = \boldsymbol{sk_i} \cdot \boldsymbol{F_j(i)}.\boldsymbol{C}$ if he receives incorrect $\boldsymbol{f_j(i)}$
  - Send $(\boldsymbol{C'}, \boldsymbol{j}, \boldsymbol{i}, \boldsymbol{p})$

Report Malicious Participant

$\boldsymbol{log_G(sk_i G) = log_{F_j(i).C}(C')}$?

- Election
  - Verify the decryption key $\boldsymbol{cpVer\left(G, pk_i, F_j(i).C, C'\right)}$
  - Decrypt $\boldsymbol{F_j(i)}$ by using $\boldsymbol{C'}$
  - Verify $\boldsymbol{f_j(i)}$ by checking $\boldsymbol{f_j(i)G = \sum_{k=0}^{t-1} A_{j,k} \cdot i^k}$ for $\boldsymbol{j} \in [\boldsymbol{1}, \boldsymbol{m}]$ and $\boldsymbol{P_j} \notin \boldsymbol{Q}$
  - Put $\boldsymbol{P_j}$ in disqualified participants set $\{\boldsymbol{P_j}\} \cup \boldsymbol{Q}$ if $\boldsymbol{f_j(i)}$ is incorrect

Vote Stage

- Set Vote Public Key
- Vote

Set Vote Public Key

- Election
  - set honest participants $\boldsymbol{P' = P - Q}$
  - If $|\mathbf{P' < t}|$, -> failed stage
  - participants’ public key $\boldsymbol{pk = pk - pk_j}$ for $\boldsymbol{j \in [1, m], P_j \in Q}$
  - Compute public key $\boldsymbol{H = \sum_{pk} pk_i}$

Vote

- Participant
  - $\mathbf{B_b}$: $\mathbf{elgamalEnc(C_j, H)}$
  - $\mathbf{sig_{hB_b}}$: $\mathbf{lrsSign(hB_b, pk, sk_i) = (u_b 1, V_b, K_b)}$
  - Send $\mathbf{B_b}$ and $\mathbf{sig_{hB_b}}$

# of ballots

- Election
  - Verify double voting $\mathbf{K_b} \in \mathbf{K}$?
  - Verify signature $\boldsymbol{lrsVer(hB_b, sig_{hB_b})}$
  - Store $\mathbf{B_b}$ and $\mathbf{sig_{hB_b}}$
  - $\{\boldsymbol{K_b}\} \cup \boldsymbol{K}$ and $\boldsymbol{b = b + 1}$

Reconstruction Stage

- Participant $\boldsymbol{P_i}$
  - $\boldsymbol{f_j(i)}$: $\boldsymbol{elgamalDec\left(F_j(i), sk_i\right)}$ for $\boldsymbol{j} \in [\mathbf{1}, \boldsymbol{m}]$ and $\boldsymbol{P_j} \notin \boldsymbol{P'}$
  - Send $\boldsymbol{f_j(i)}$ for $\boldsymbol{j} \in [\mathbf{1}, \boldsymbol{m}]$ and $\boldsymbol{P_j} \notin \boldsymbol{P'}$
- Election
  - Set $\boldsymbol{f(i)} = \mathbf{0}$
  - for $\boldsymbol{j} \in [\mathbf{1}, \boldsymbol{m}]$ and $\boldsymbol{P_j} \notin \boldsymbol{P}$
    - $\boldsymbol{f(i)}$ += $\boldsymbol{f_j(i)}$ if $\boldsymbol{f_j(i)G} = \sum_{k=0}^{t-1} \boldsymbol{A_{j,k}} \cdot \boldsymbol{i^k}$
    - Otherwise, -> error
  - Store $\boldsymbol{f(i)}$

Result Stage

- Recover Private Key
- Tally the Ballots

Recover Private Key

- T: a set of $\boldsymbol{t}$ number of participants who submitted their shares
- $\boldsymbol{S = \sum_{i \in T} f(i) \lambda_i(0)}$, **where** $\boldsymbol{\lambda_i(0) = \prod_{j \in T, i \neq j} \frac{j}{j-i} \pmod{n}}$
- Send $\boldsymbol{S}$

Tally the Ballots

- If $|\boldsymbol{f}(\boldsymbol{i}) < \mathbf{t}|$, -> failed stage
- Verify Private Key $\boldsymbol{H} = \boldsymbol{S} \cdot \boldsymbol{G}$?
- For each $\boldsymbol{B_i}$,
  - $\boldsymbol{C_j} = \boldsymbol{elgamalDec}(\boldsymbol{B_i}, \boldsymbol{S})$
  - $\boldsymbol{C_j}.\boldsymbol{votecount} \mathrel{+}= \mathbf{1}$ if $\boldsymbol{C_j} \in [\mathbf{1}, \boldsymbol{d}]$

DEMO

DEMO

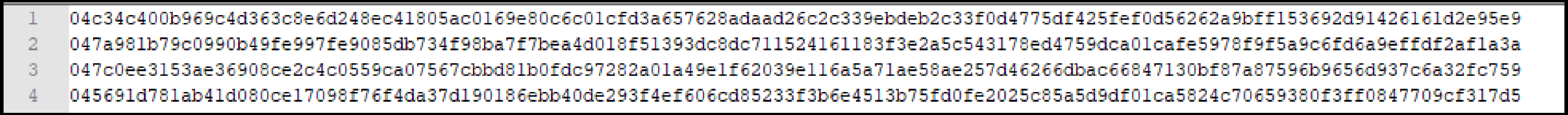
- Timers off
- isRegOn: True
- Participants: P_0, P_1, P_2, P_3
- # of register: 1
  - Become P_4
- Candidates: a, b
- minShares: 2

DEMO

- Non-contributed Participant: $\boldsymbol{P_0}$
- Malicious Participant: $\boldsymbol{P_2}$
- Vote: $\boldsymbol{P_1}$
- Submit Shares: $\boldsymbol{P_1, P_3}$

DEMO

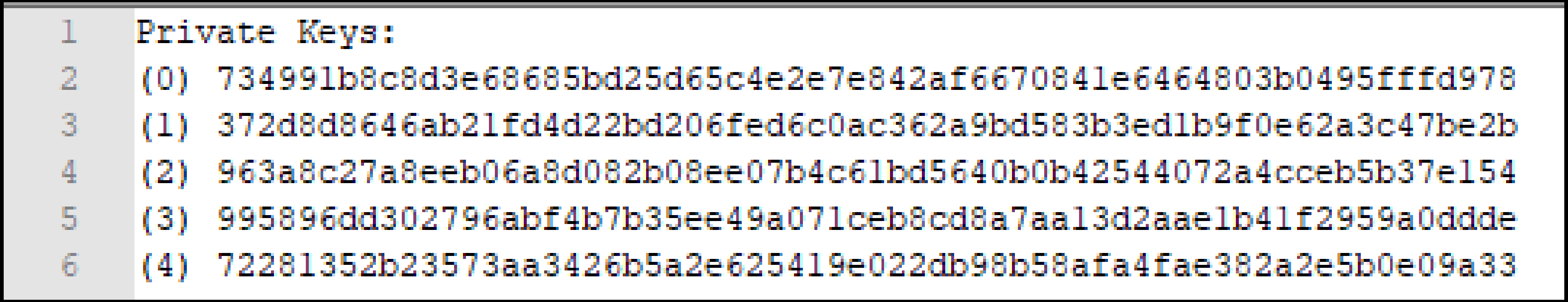
- Participant Public Keys



```
04c34c400b969c4d363c8e6d248ec41805ac0169e80c6c01cfd3a657628adaad26c2c339ebdeb2c33f0d4775df425fef0d56262a9bff153692d91426161d2e95e9
047a981b79c0990b49fe997fe9085db734f98ba7f7bea4d018f51393dc8dc711524161183f3e2a5c543178ed4759dca01cafe5978f9f5a9c6fd6a9effdf2af1a3a
047c0ee3153ae36908ce2c4c0559ca07567cbbd81b0fdc97282a01a49e1f62039e116a5a71ae58ae257d46266dbac66847130bf87a87596b9656d937c6a32fc759
045691d781ab41d080ce17098f76f4da37d190186ebb40de293f4ef606cd85233f3b6e4513b75fd0fe2025c85a5d9df01ca5824c70659380f3ff0847709cf317d5
```

DEMO

- Participant Private Keys



```
Private Keys:
(0) 734991b8c8d3e68685bd25d65c4e2e7e842af6670841e6464803b0495fffd978
(1) 372d8d8646ab21fd4d22bd206fed6c0ac362a9bd583b3ed1b9f0e62a3c47be2b
(2) 963a8c27a8eeb06a8d082b08ee07b4c61bd5640b0b42544072a4cceb5b37e154
(3) 995896dd302796abf4b7b35ee49a071ceb8cd8a7aa13d2aae1b41f2959a0ddde
(4) 72281352b23573aa3426b5a2e625419e022db98b58afa4fae382a2e5b0e09a33
```