Blockchains & Distributed Ledgers

Lecture 09

Dimitris Karakostas

Slide credits: DK, Aggelos Kiayias

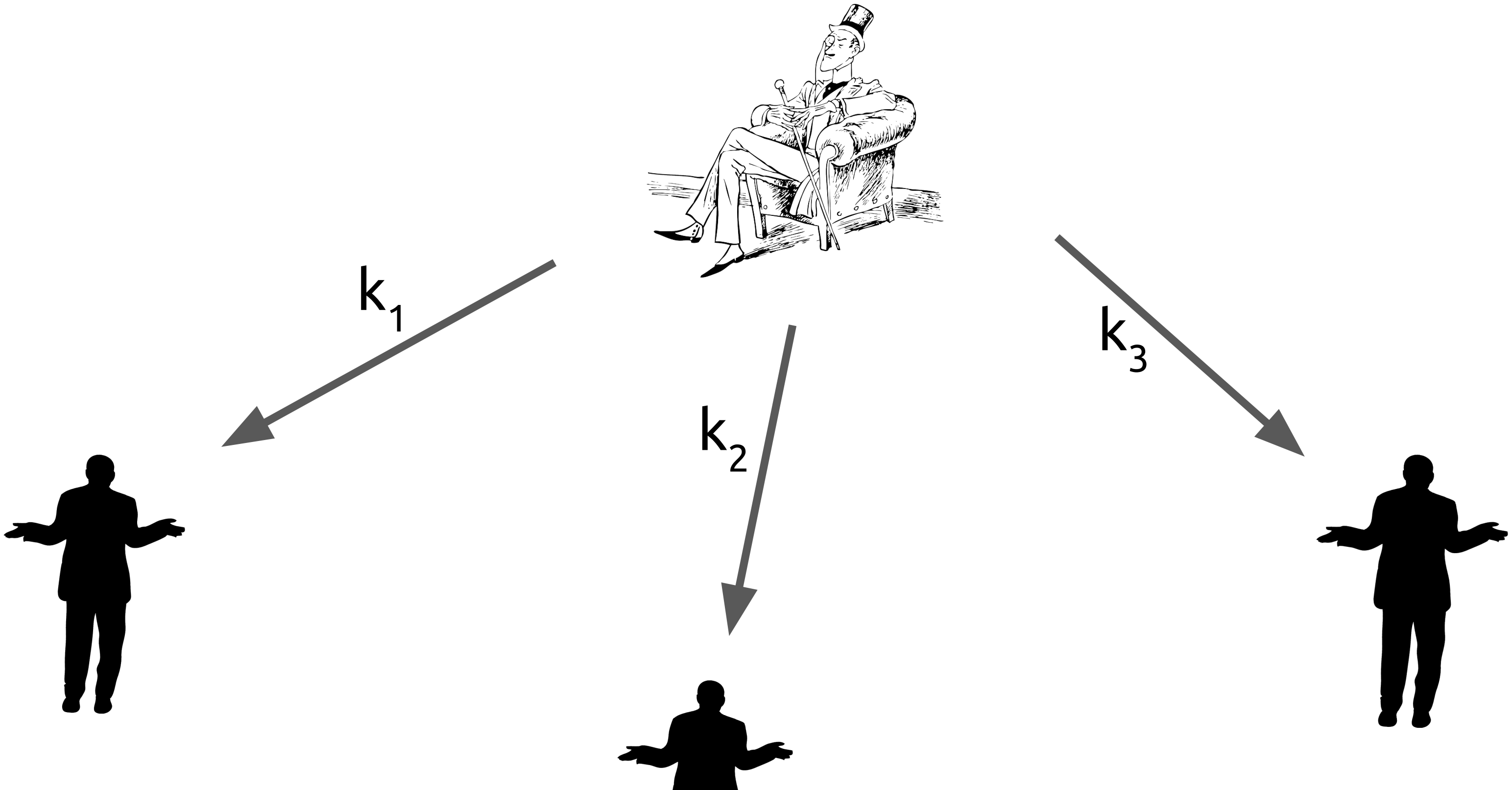
Security critical computations

- How to obtain the output of a security critical computation
- Deterministic with public inputs?
  - Repeat multiple times and consensus can be reached about its output
  - Example: blockchain systems with smart contracts
- What if it is probabilistic with public inputs?
  - Coin flipping protocol
- What if it uses private data?
  - Secure Multiparty Computation (MPC)

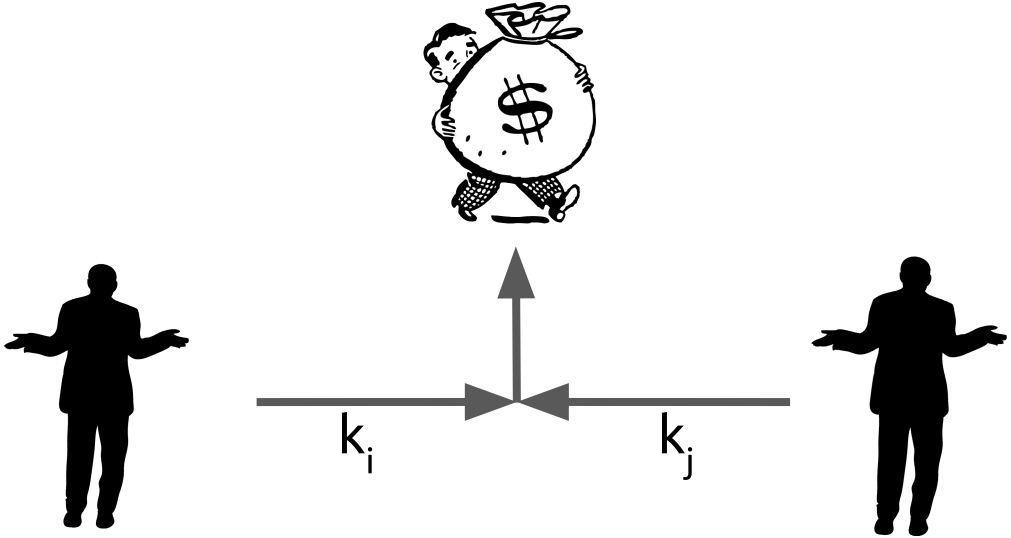
Secure Multiparty Computation and Applications

- Sharing responsibility for signatures and cryptographic keys
  - Secret sharing
- Security critical computations
  - Coin flipping and verifiable secret-sharing
  - Secure multiparty computation (MPC)
- Fair swaps and fair MPC

Secret sharing



k1
k2
k3



$i \neq j$; $i,j \in$
$\{1,2,3\}$

Overarching question

- How to protect security critical operations?
- Key idea: share responsibility and somehow tolerate faulty participants
  - Cryptographic keys?
  - Cryptocurrency addresses?
  - Computations?
  - What about computations on private data?

Multi-sig transactions

- Multi-sig: a tx that can be redeemed if n parties sign it
- A payment to a script (P2SH) can facilitate a multi-signature transaction

```
scriptPubKey: OP_HASH160 <redeemscriptHash> OP_EQUAL
```

```
scriptSig: OP_0 <sig_Ai> … <sig_An> <redeemscript>
```

```
redeemscript = OP_m <A1 pubkey> <A2 pubkey>… <An pubkey>
<OP_n> <OP_CHECKMULTISIG>
```

Secret-Sharing

Main question:

- How to share a secret *s* to *n* shareholders so that:
  - Any subset including *t* of them can <u>recover</u> the secret
  - Any subset including *less than t* of them knows <u>nothing</u> about the secret

- Relative questions:
  - Can we solve this for any *n* and *t* <= *n*?
  - What is the relation between the size of *s* and the size of each share?

Finite fields

- Finite *sets* equipped with *two operations*, behaving similarly to addition and multiplication over the real numbers (which is an infinite field)
- Finite fields exist with number of elements equal to p^k, for:
  - any prime number p
  - any positive integer k

Example. A binary finite field {0, 1} with:

+	0	1
0	0	1
1	1	0

(a+b) mod 2

*	0	1
0	0	0
1	0	1

(a*b) mod 2

Secret-Sharing over a finite field

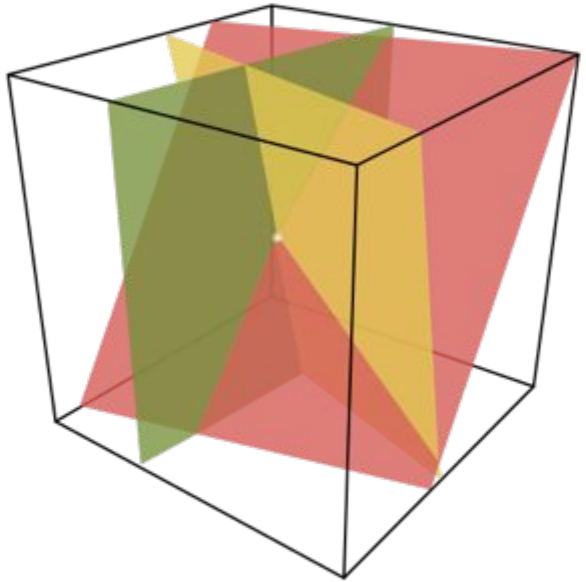
- Consider a secret x and *N* random values, subject to the constraint:

$$\sum_{i=1}^{N} x_i = x \quad \text{(over a finite field)}$$

- This is called (additive) secret-sharing
- Knowledge of *any* N-1 values cannot be used to infer any information about *x*

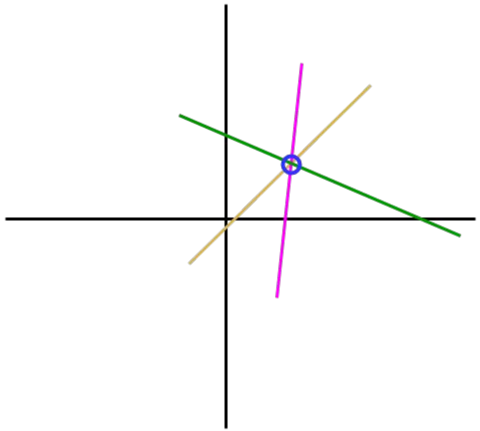
Analysis

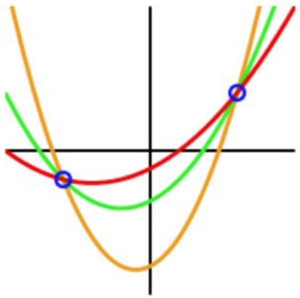
- Example: binary field
- If you hold only *N-1* values $[x_2, \ldots, x_N]$:
  - Two unknowns: x_1, s
  - One equation: $x_1 + x_2 + \ldots + x_N = s$
- *s* cannot be undetermined
  - $s = 0 + x_2 + \ldots + x_N$ (if $x_1 = 0$)
  - $s = 1 + x_2 + \ldots + x_N$ (if $x_1 = 1$)



Generalisation t-out-of-n

- Consider a polynomial of degree *d*: $\boldsymbol{p(x) = a_0 + a_1x + \ldots + a_dx^d}$
- Any *d+1* points of the polynomial completely determine it
- With *d* or less points, at least one degree of freedom remains
    - *p* cannot be fully determined
- We can use that idea to solve secret-sharing for any t, n





Example

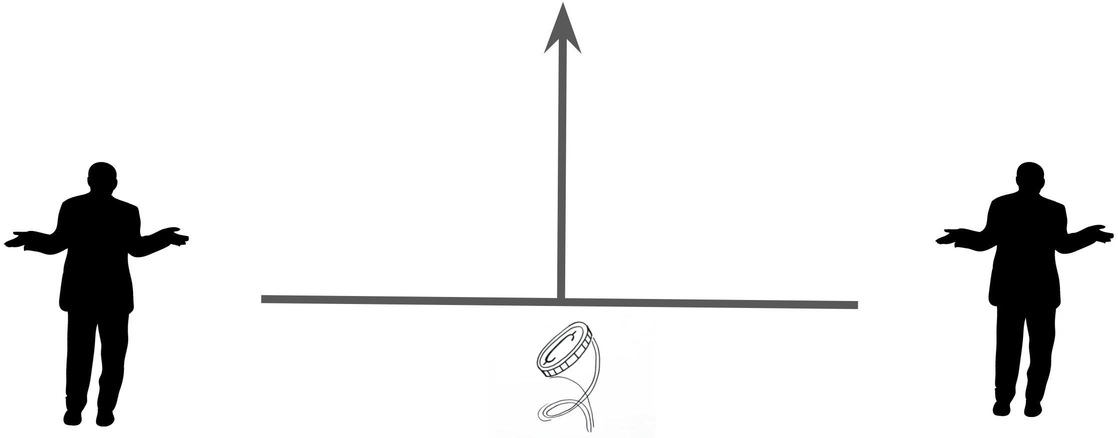
- 5 parties
- Polynomials of degree 2
- Any three parties (who hold 3 points) can interpolate such polynomials
- Any two parties have no information about the shared secret

Secret-sharing cryptographic keys

- Using polynomial secret-sharing, a cryptographic key can be split to multiple shareholders
    - Each shareholder gets a point on the plane
    - The secret/key is the solution to the polynomial problem
- Additional points to consider:
    - How should the value of *t* be determined:
        - in comparison to *d*?
        - in comparison to n?
    - To engage in the cryptographic operation, is it necessary to reconstruct the original key?
    - How to accomodate an evolving set of shareholders?

Distributed Randomness Generation

$$b \xleftarrow{\$} \{0, 1\}$$



Application: coin-flipping

- Alice and Bob want to flip a coin remotely
  - output a bit uniformly at random
- Alice doesn't trust Bob and vice versa
  - neither Alice nor Bob should be able to bias the bit choice

Application: coin-flipping

- Alice and Bob want to flip a coin remotely
  - output a bit uniformly at random
- Alice doesn't trust Bob and vice versa
  - neither Alice nor Bob should be able to bias the bit choice
- Solution:
  - Alice commits to a random coin
  - Bob commits to a random coin
  - Alice and Bob open the commitments
  - Output = XOR of (committed) values
- Consider:
  - Can the situation be improved in an *N* party coin flip?
  - What about when >N/2 parties are honest?
  - How do you deal with (selective) aborts?

A first step towards multi-party coin flipping

- Each player commits to their coin (publicly)
- Each player publishes a secret-sharing of the opening to their commitment
  - Any subset of at least (N/2 + 1) players can reconstruct the opening
  - Shares should be encrypted with the respective public-keys of the parties
- If some parties abort the protocol: assuming that a subset of $>N/2$ parties continue, they can recover the share and terminate
- Any number of parties up to N/2 cannot gain any advantage over the honest parties

What if some parties announce incorrect shares?

- A secret cannot be retrieved from incorrect shares
- Selective aborts possible, as remaining parties cannot reconstruct the secret
- Possible solution: require that all commitments open at the end irrespectively of aborts
  - deviating players will be caught, but still have the option to selectively abort if they wish
  - other parties will only know of the abort when it is too late
- One possible approach: issue monetary penalties to those that abort

Publicly Verifiable Secret-sharing (PVSS)

- The dealer creates shares that are distributed in encrypted form
- The shares can be **publicly verified** as correct
- Verifiability should not leak information about the secret

- PVSS enables parties to detect improper share distribution at the onset
- Protocol can still be aborted, but any abort would be independent of the (random) coin!

PVSS Design Challenges

- Assuming:

$$\sum_{i=1}^{N} x_i = x \qquad \psi_i = \mathcal{E}_i(x_i)$$

$$\psi = \mathbf{Com}(x)$$

- Verify that the value committed in ψ_i satisfies the equation w.r.t. the values encrypted in ψ
- This problem can be solved using a **zero-knowledge proof**

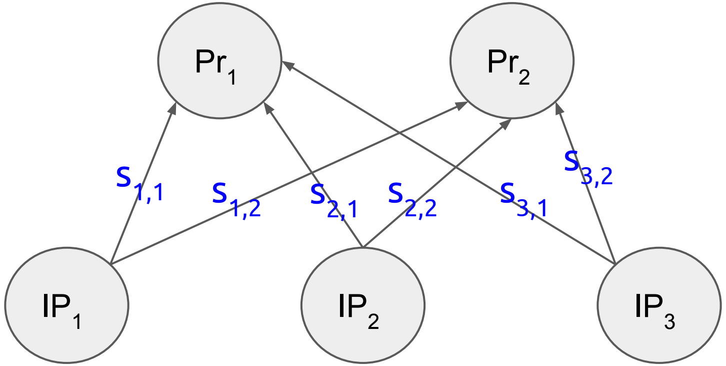
Secure MPC

Secure Multiparty Computation

- (Secure) Multiparty Computation (MPC)
- Parameterized by function $f(.)$
- A set of n parties $\mathrm{P_i}$ contribute inputs $x_1, x_2, \ldots, x_n$
- At the end of the protocol they compute $f(x_1, x_2, \ldots, x_n)$
    - Everyone receives output $f(x_1, x_2, \ldots, x_n)$
    - No party except $\mathrm{P_i}$ obtains information about x_i

MPC Construction Idea

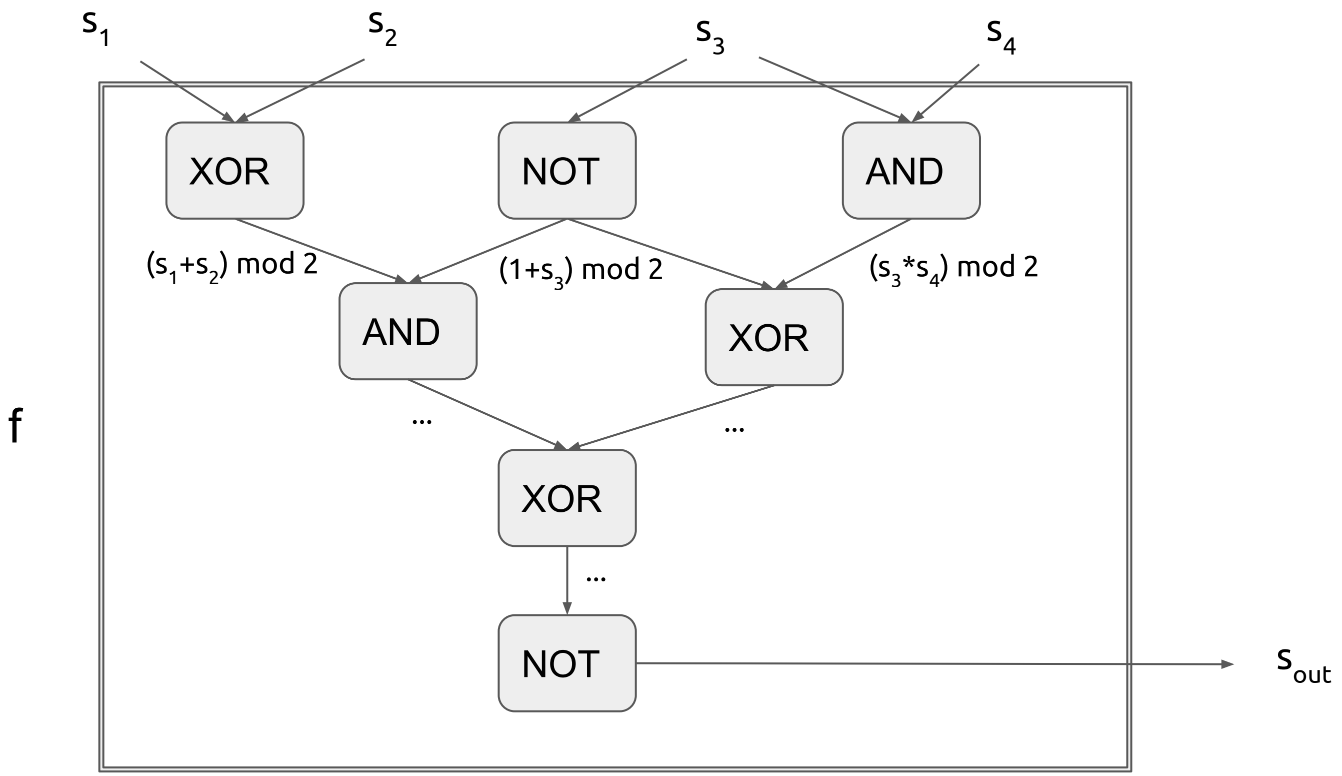
- Consider three roles
  - Input providers
  - Processors
  - Output-receivers
- Input providers secret-share their input to the processors
  - Additive secret-sharing



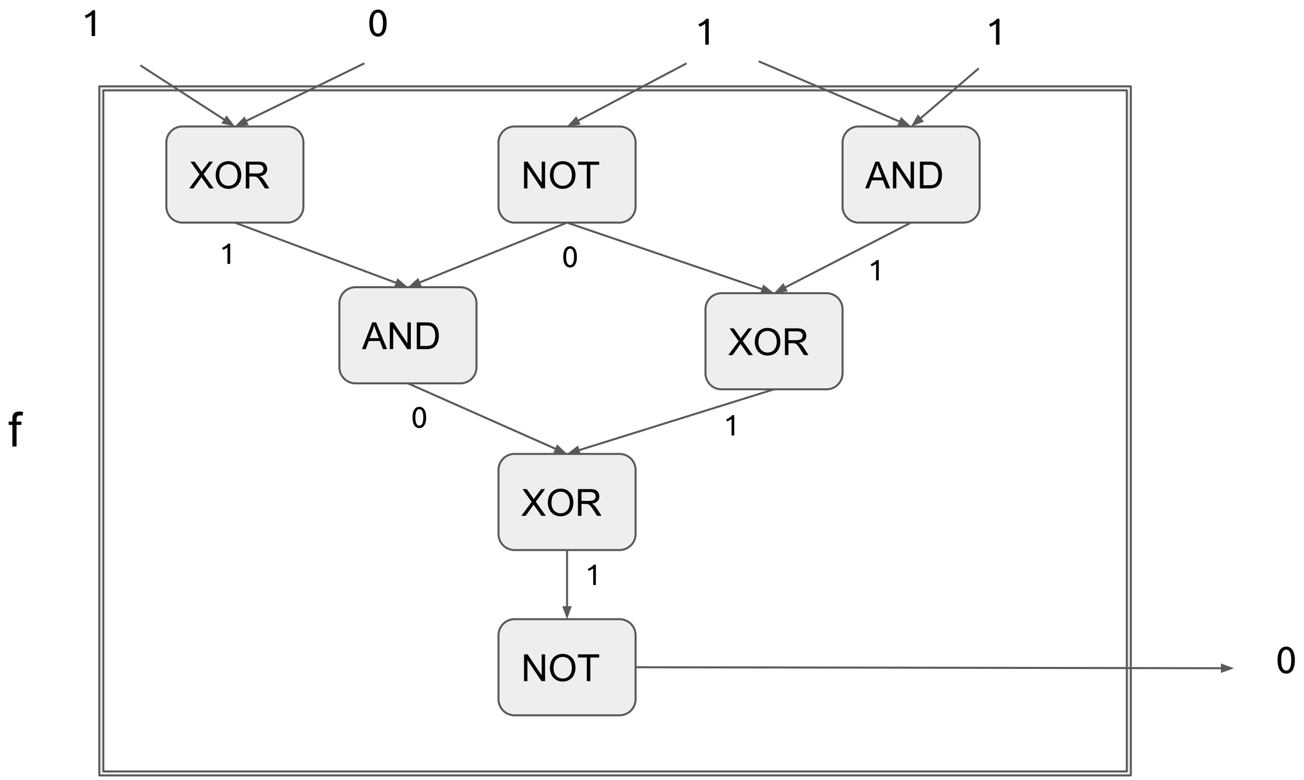
MPC Construction Idea

- Any function *f* can be expressed as a Boolean circuit
  - Fixed-size input
  - Upper-bound on number of steps (circuit depth)
  - Example: any boolean function can be implemented as a combination of NAND gates
- XOR, AND, NOT gates
- Arithmetic representation of gates
  - AND: Input: a, b; Output: (a*b) mod 2
  - XOR: Input: a,b; Output: (a+b) mod 2
  - NOT: Input: a; Output: (1+a) mod 2
- Each processor executes the circuit with their shares as input
  - How to implement the gates s.t. operations on shares, when combined, produce the correct aggregate output?

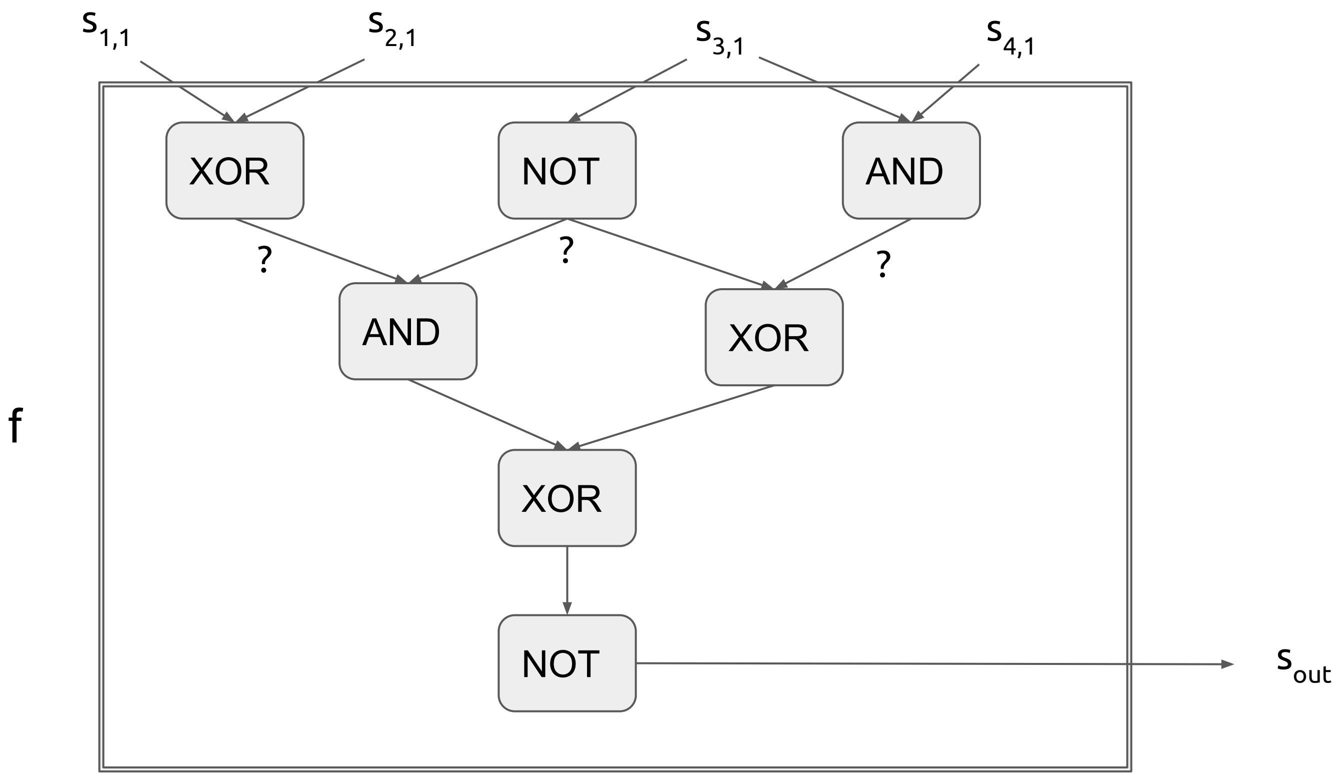
MPC Construction Idea, Example



MPC Construction Idea, Example



MPC Construction Idea, Example



MPC Construction Idea

NOT GATE

- Suppose *m* parties hold shares of two inputs to a NOT gate.

$$[a] = \langle a_1, \ldots, a_m \rangle$$

- How do they calculate shares of the output of the NOT gate?

$$[\overline{a}] = \langle 1 + a_1 \bmod 2, a_2, \ldots, a_m \rangle$$

MPC Construction Idea

XOR GATE

- Suppose *m* parties hold shares of two inputs to an XOR gate.

$$[a], [b] = \langle a_1, \ldots, a_m \rangle, \langle b_1, \ldots, b_m \rangle$$

- How do they calculate shares of the output of the XOR gate?

$$[a] + [b] \bmod 2$$

MPC Construction Idea

- Suppose *m* parties hold shares of two inputs to an AND gate.

AND GATE

$$[a], [b] = \langle a_1, \ldots, a_m \rangle, \langle b_1, \ldots, b_m \rangle$$

- How do they calculate shares of the output of the AND gate?

$$[a] \cdot [b] = \langle a_1 b_1 \bmod 2, \ldots, a_m b_m \bmod 2 \rangle$$

but we want: $s_1 + \ldots + s_m = (\sum_{i=1}^{m} a_i)(\sum_{i=1}^{m} b_i)$

MPC Construction Idea

- A Beaver triple is an initial secret-sharing of random values

$$x \cdot y = z$$

$$[x] = \langle x_1, \ldots, x_m \rangle, [y] = \langle y_1, \ldots, y_m \rangle, [z] = \langle z_1, \ldots, z_m \rangle$$

AND GATE :

publish $d_i = a_i - x_i$ reconstruct d, e

$e_i = b_i - y_i$

define $s_i = de \quad + dy_i + ex_i + z_i$ share calculation

$$\sum s_i = de + d \sum y_i + e \sum x_i + xy$$

(assuming m is odd)

$$= de + dy + ex + xy = (a - x)(b - y) + (a - x)y + (b - y)x + xy$$

$$= ab$$

Constructing Beaver Triples

- The above construction idea requires the setup of all servers with a sufficient number of Beaver triples (how many?)
- Constructing Beaver triples can be done via special-purpose cryptographic protocols

MPC strengths and weaknesses

- Possible to compute any function *f* privately on parties' inputs
- Unless *honest majority* is present, there is no way to provide:
  - fairness: either all parties learn the output or none
  - guaranteed output delivery

Fairness

Workarounds for fairness

- Optimistic fairness (by involving a third party):
  - The protocol is basically not fair
  - A third party is guaranteed to be able to engage and amend the execution in case of deviation
- Gradual/timed release:
  - Protocols engage in many rounds
  - Parties gradually come closer to computing the output
  - “gradual closeness” can be measured in terms of:
    - probability of guessing the output
    - number of computational steps remaining to compute the output
  - Example:
    - At each round $l = 1,...,n$ the two parties can compute the output in 2^{n-l} steps
    - If a party aborts the interaction, the other party will be 2 times more steps “behind” in the calculation of the output

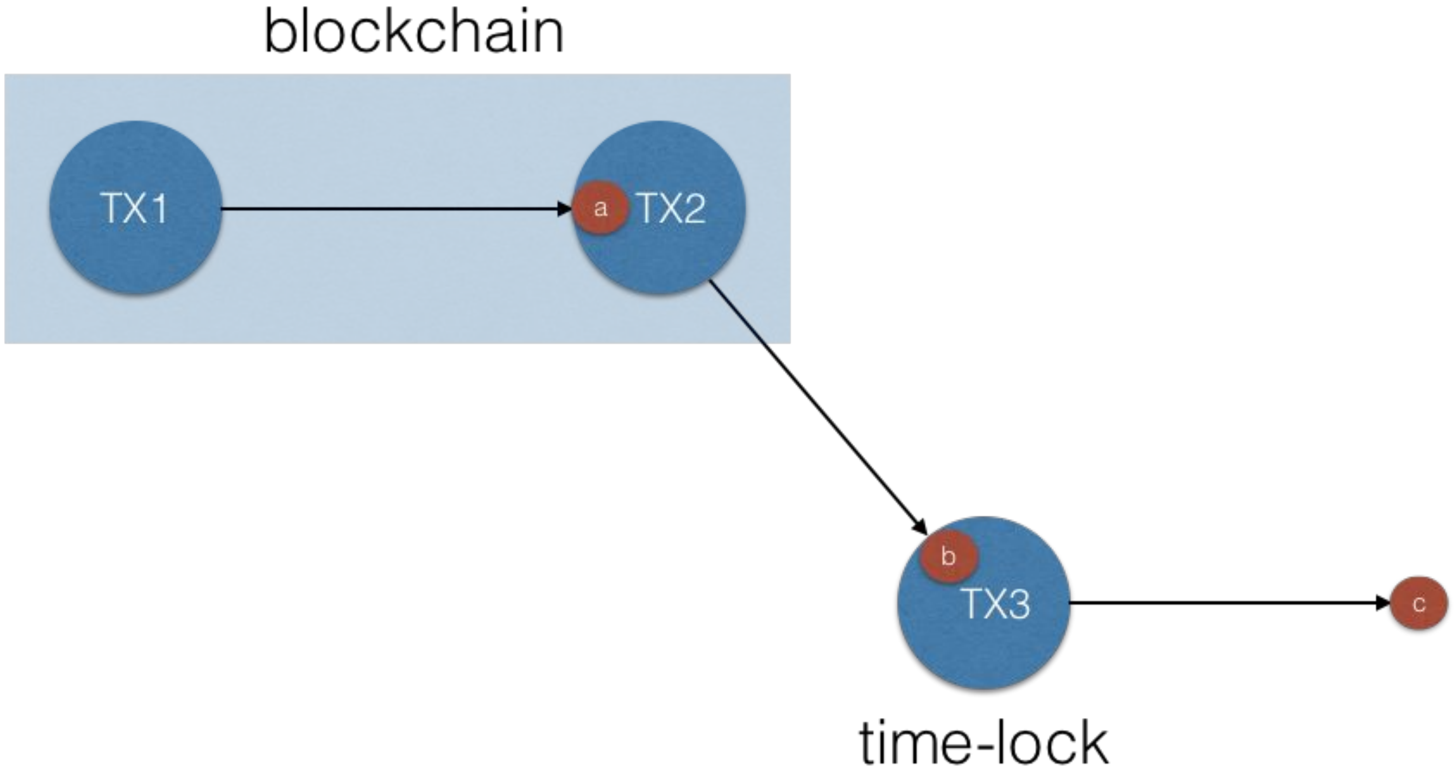
Using a blockchain

- Along the lines of optimistic fairness, but substituting the trusted third party with the blockchain
- How is that possible?
    - Blockchain cannot keep secrets
    - Rationale: penalize parties that deviate from the protocol

Basic tool: time-lock transactions

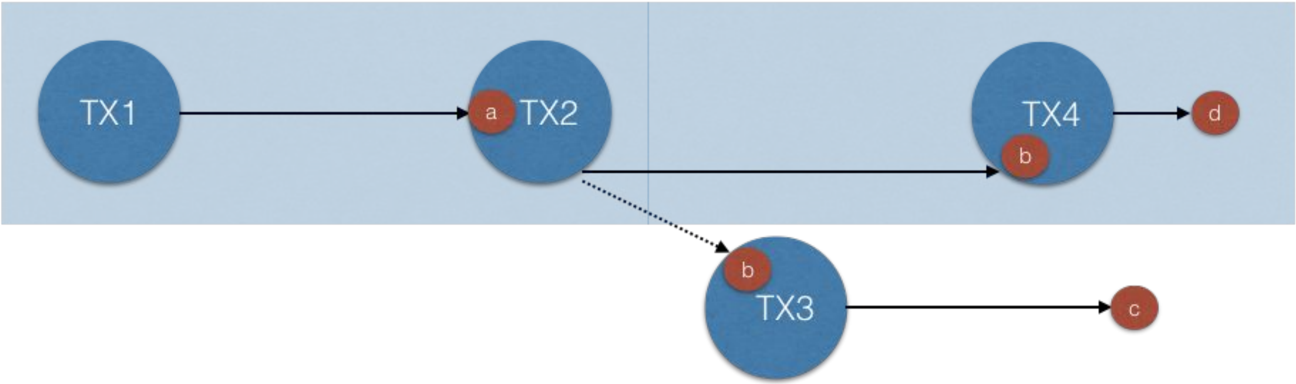
- Time-lock transactions
  - part of transaction data
  - specifies the earliest time that a transaction can be included in a block
- Key observation: if a conflicting transaction has already being included in the ledger, the time-lock transaction will be rejected

Time-lock example



Time-lock example

OR



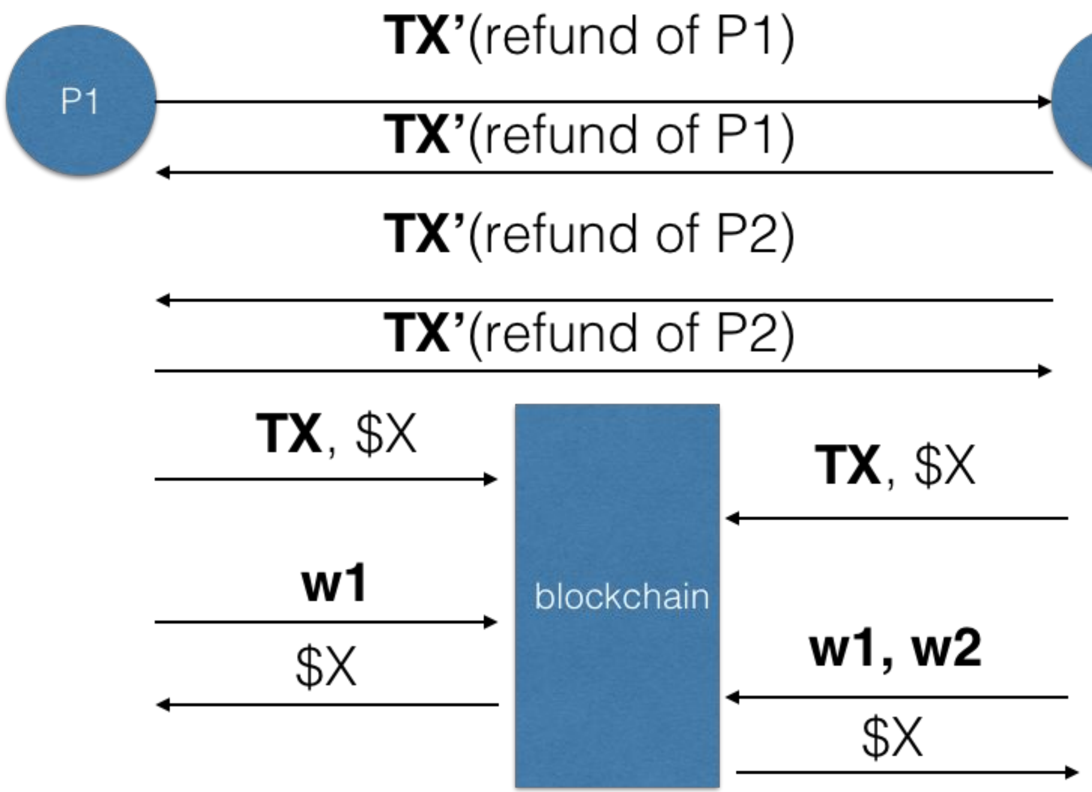
Fair swap of values using time-locks, Setup

- P_1 holds $\mathbf{w_1}$**,** $\mathbf{h_2=H(w_2)}$
- P_2 holds $\mathbf{w_2}$**,** $\mathbf{h_1=H(w_1)}$
- *They want to exchange* w_1, w_2

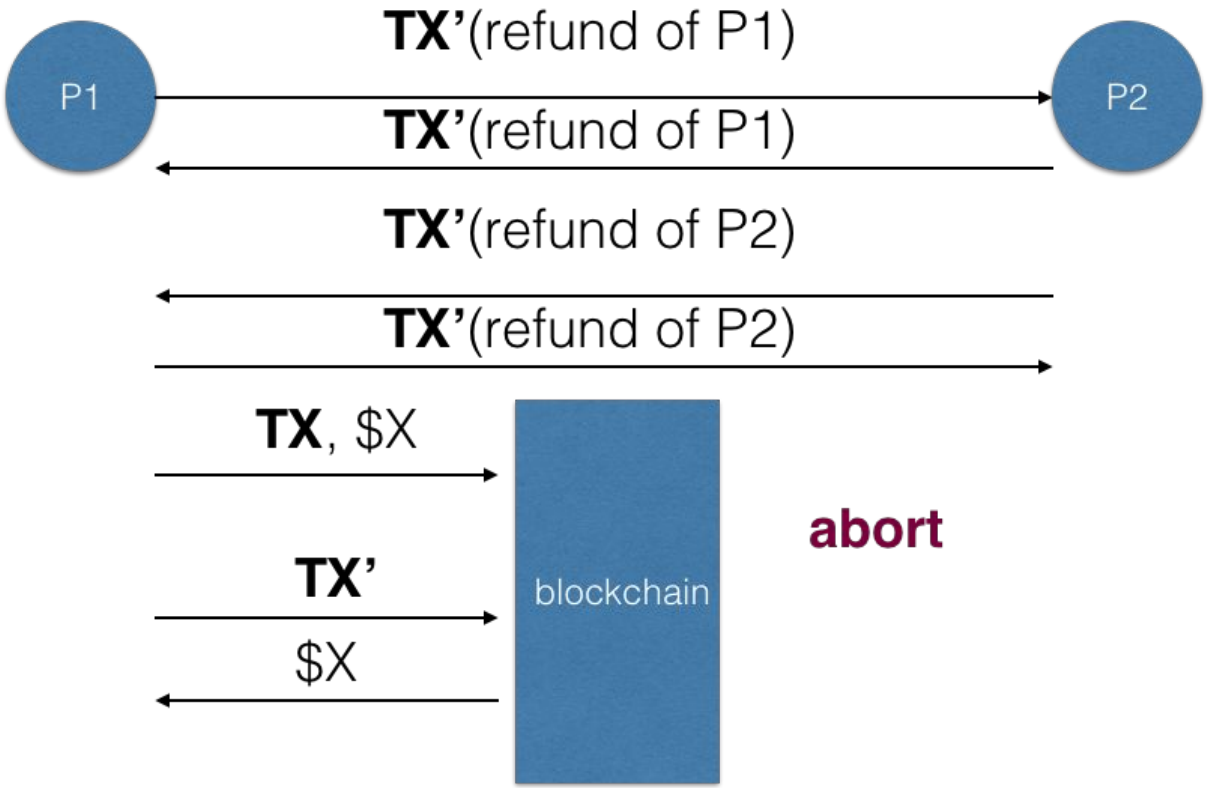
Fair swap of values using time-locks, Execution

- P_1:
  - Creates a P2SH transaction TX for $X provided that:
    1. (P_1 and P_2 sign, as 2-out-of-2 multisignature) *or*
    2. (P_2 signs and reveals $\mathbf{w_1}$, $\mathbf{w_2}$, s.t. $H(\mathbf{w_1}) = \mathbf{h_1}$ and $H(\mathbf{w_2}) = \mathbf{h_2}$)
  - Creates a P2PKH transaction TX' that spends the output of TX with a time-lock in the near future
  - Sends TX' to P_2 to sign it (P_2 does not see TX, only the tx id is needed to refer to it)
- P_2 acts in the same way:
  - Create a TX that can be redeemed via (2-out-of-2 multisig) or (P_1 signs and reveals $\mathbf{w_1}$, s.t. $H(\mathbf{w_1})=\mathbf{h_1}$)
  - Create a corresponding time-locked TX' and send to P_1 to sign
- Completion:
  - P_1 publishes its TX, so P_2 can redeem $X by revealing $\mathbf{w_1}$, $\mathbf{w_2}$
  - P_2 publishes its TX, so P_1 can redeem $X by revealing $\mathbf{w_1}$
  - P_1 reveals $\mathbf{w_1}$ and redeems $X (from P_2's TX)
  - P_2 reveals $\mathbf{w_1}$, $\mathbf{w_2}$ and redeems $X (from P_1's TX)
- If either party aborts, the other can claim $X (from their TX) after time-lock fires, by publishing their TX'

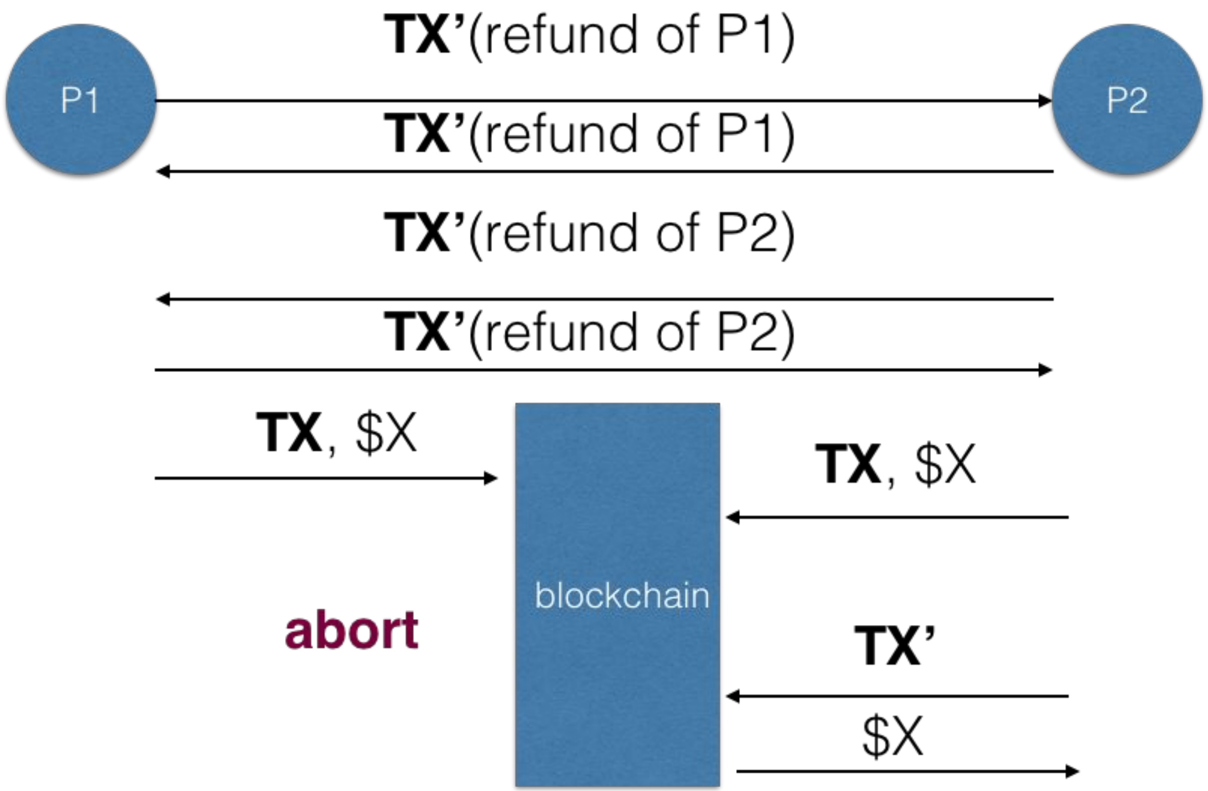
Fair swap of values using time-locks



Fair swap of values using time-locks



Fair swap of values using time-locks



Fair swap of values using time-locks, Notes

- If P_1's TX could be redeemed by "$H(\mathbf{w_2}) = \mathbf{h_2}$ and P_2 signs it":
  - P_2 could reveal $\mathbf{w_2}$ and obtain payment of \$X, without publishing its own TX transaction
  - P_1 would obtain the output $\mathbf{w_2}$ but lose \$X
  - (note that we cannot ensure that the TX transactions will appear concurrently in the blockchain)
- If a multisig was not used for the refunds, a player could:
  - Submit its value
  - Rush to obtain its refund, invalidating the TX payment of the other player
- The time-lock for P_1 should be less than that for P_2; if equal, P_1 could:
  - Wait for the very last minute to reveal $\mathbf{w_1}$
  - Hope that time-lock fires before P_2 can publish $\mathbf{w_2}$ on the chain
  - Claim \$X even if P_2 tries to act honestly (and reveals $\mathbf{w_1}$ out of time)

Fair Computation

- The two parties use MPC to compute a secret sharing of the output of the computation
  - **$w_1 + w_2$ = MPC_output**
- Subsequently parties do a fair swap of values, to obtain the MPC_output:
  - If a party aborts, the other will be compensated

N-party ladder construction, I

- Uses N-out-of-N multisig for refunds
- P_N can redeem \$X from each player if it reveals $\mathbf{w_1}$, $\mathbf{w_2}$, …, $\mathbf{w_N}$ (i.e., the N-1 parties prepare these "roof" **TX** transactions)
- For i = 1, …, N-1, player P_{N-i} can redeem from player P_{N-i+1} an amount equal to \$X(N-i) if it reveals $\mathbf{w_1, w_2, \ldots, w_{N-i}}$ (the N-1 parties also prepare these "ladder" **TX** transactions)
- Redeeming follows the sequence P_1, P_2, …, P_N

N-party ladder construction, II

- P_1 will redeem \$X from P_2 for publishing $\mathbf{w_1}$
- P_2 will redeem \$2X from P_3 for publishing $\mathbf{w_1, w_2}$
- …
- P_{N-1} will redeem \$(N-1)X from P_N for publishing $\mathbf{w_1, w_2, \ldots, w_{N-1}}$
- P_N will redeem \$X from each of $P_1, \ldots, P_{N-1}$ for publishing $\mathbf{w_1, w_2, \ldots, w_N}$