Secure Multiparty Computation and Applications

Introduction to Blockchain Science and Engineering

Aggelos Kiayias

Dionysis Zindros, Christos Nasikas

Secure multiparty Computation and Applications

- Sharing responsibility for accounts and keys
  - Secret Sharing
  - Verifiable secret sharing
- Secure multiparty computation (MPC)
- Fair swaps and MPC

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

- [Recall]
- A payment to a script (P2SH) can be used to facilitate a multi signature transaction.
- m-out-of-n multisig script:
  - <OP_m> <A1 pubkey> <A2 pubkey>… <An pubkey> <OP_n> <OP_CHECKMULTISIG>

Mutli-sig Transactions, II

```
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>
```

Sharing Secret Keys

- Structure of secret-keys in bitcoin.
- Uses the digital signature algorithm (DSA).
  - Standardized in the digital signature standard by National Institute of Standards and Technology (NIST).
- Alternative algorithms: Schnorr

Finite Cyclic Groups

$$\langle g^0, g^1, \ldots, g^{q-1} \rangle$$

$$g^q = 1$$

$$g^x = g^{x \bmod q}$$

$$g^x g^y = g^{x+y}$$

Can be constructed using modulo operations over the integers or over elliptic curves.

Exponent operations

$$g^x = g^y$$

$$\gcd(q, y) = 1 \Longrightarrow \exists y^{-1} \in \{1, \ldots, q-1\} : g^{xy^{-1}} = g$$

(Computable via the Extended Euclidian Algorithm)

Typical choice: choose q to be a prime number

DSA

pk ↓ $y = g^x$ sk ↓

Sign

Let H be the hashing function and m the message:

- Generate a random per-message value k where $1 < k < q$
- Calculate $r = \left(g^k \bmod p\right) \bmod q$
- In the unlikely case that $r = 0$, start again with a different random k
- Calculate $s = k^{-1}\left(H(m) + xr\right) \bmod q$
- In the unlikely case that $s = 0$, start again with a different random k
- The signature is (r, s)

Verify

- Reject the signature if $0 < r < q$ or $0 < s < q$ is not satisfied.
- Calculate $w = s^{-1} \bmod q$
- Calculate $u_1 = H(m) \cdot w \bmod q$
- Calculate $u_2 = r \cdot w \bmod q$
- Calculate $v = (g^{u_1} y^{u_2} \bmod p) \bmod q$
- The signature is invalid unless $v = r$

observe $g^{u_1} y^{u_2} = g^{H(m)w} y^{rw} = g^{(H(m)+xr)s^{-1}} = g^k$

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

Finite Fields

- A field is a set equipped with two operations, addition and multiplication, that behave in a way that abstracts the real numbers.
- **Finite fields** are particularly useful in computer science as (i) their elements accept a succinct representation, (ii) we can analyze their properties using discrete mathematics.

Finite fields and cyclic groups

$$\langle g^0, g^1, \ldots, g^{q-1} \rangle$$

If the order q of the group is prime
then exponent arithmetic is that of
a finite field (prime order field q)

Example. $\langle g^0, g^1, g^2, g^3, g^4 \rangle$ order 5 cyclic group

$$g^5 = 1$$

$$(g^2)^3 = g$$

$$(g^4)^4 = g$$

Secret-Sharing

- Consider random *N* values subject to the constraint

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

- This is called a secret-sharing.
- Observe knowledge of any *N*-1 values is not helpful in any way to infer information about *x*

Analysis

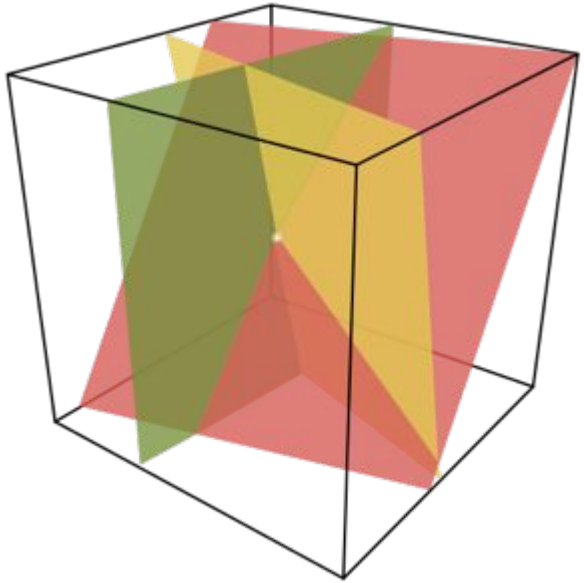
- Imagine you hold *N*-1 values.
- Example binary field.

$$x_1 + x_2 + \ldots + x_N = s$$

or

$$s = 0 + x_2 + \ldots + x_N$$

$$s = 1 + x_2 + \ldots + x_N$$



Sharing a secret-key

- Observe that DSA (and also Schnorr) keys are of the form $g^x = y$

N parties can share a key so that :

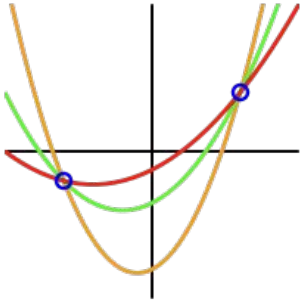
Key generation procedure: $\sum x_i = x \bmod q$

each party publishes $y_i = g^{x_i}$

public-key is set to: $y = \prod y_i$

Generalisation t-out-of-n

- Recall the notion of a polynomial.
- $p(x) = a_0 + a_1 x + \ldots + a_d x^d$



Any *d+1* points of the polynomial completely determine it. Given up to *d* points, at least one degree of freedom remains and it remains undetermined.

Security critical computations

How to obtain the output of a security critical computation.

1. Deterministic with public inputs? [it can be repeated multiple times]
2. What if it is probabilistic with public inputs?
    a. Coin flipping protocol.
3. What if it uses private data?
    a. Secure multiparty computation

Application: coin-flipping

- Recall the classic two-party coin flipping protocol: Alice commits to Bob a random coin, Bob sends her a random coin and they return the XOR of them (with Alice opening her coin).
- Consider: how do you deal with aborts?
- Can the situation be improved in an *N* party coin flip? what about when >N/2 parties are honest?

A first step towards coin flipping

- Players commit their coin publicly and also include a secret-sharing of the opening of the commitment so that any subset of players with cardinality > *N*/2 can reconstruct the opening. Shares should be encrypted with the respective public-keys of the parties.
- Thus if some parties abort the protocol, assuming that a subset of > *N*/2 parties continue they can recover the share and terminate.
- At the same time any number of parties up to N/2 are unable to gain any advantage against the honest parties (they will be one party short).

Example

- Consider 5 parties. We choose polynomials of degree 2.
  - Any three parties, holding 3 points, are able to interpolate such polynomials.
  - Any two parties, have no information about the shared secret.

A critical issue that remains

- What happens when some parties deliver incorrect shares?
  - Such parties that selectively abort, will cause remaining parties to be unable to reconstruct the secret.
  - Possible solution: have all commitments open at the end irrespectively of aborts.
    - Still unclear how this helps: deviating players will be caught but they still have the option to abort execution if they wish (and other parties will only know this when it is too late).

Publicly verifiable secret-sharing

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

Employing PVSS

- Using PVSS enables parties to detect improper share distribution at the onset.
- Protocol can still be aborted, but any abort would be independent of the coin.
  - (this cannot be avoided anyway).

PVSS Design Challenges

- Assuming $\sum_{i=1}^{N} x_i = x$ $\psi_i = \mathcal{E}_i(x_i)$ $\psi = \mathbf{Com}(x)$
- Verify that the value committed in ψ_i satisfies the equation with respect to the values encrypted in ψ

The cryptographic tool that solves the above problem is called a **zero-knowledge proof**.

Secure Multiparty Computation

- (Secure) Multiparty Computation (MPC)
  - Parameterized by function *f(.)*
  - A set of *n* parties contribute inputs *x1, x2, …, xn*
  - At the end of the protocol they compute *f(x1, x2, …, xn)*

MPC Construction Idea

- Consider three roles:
  - Input-providers, Processors, Output-receivers
- Input providers secret-share their input to processors
  - Secret-sharing:

Additive Secret Sharing

$$s_1 + s_2 + \ldots + s_m = x \bmod P$$

MPC Construction Idea, II

Represent function f as Boolean circuit, e.g., XOR, AND, NOT

and arithmetize it!

Gates

Addition

Input	a, b
Output	$(a + b) \bmod 2$

Multiplication

Input	a, b
Output	$(a \cdot b) \bmod 2$

NOT

Input	a
Output	$(1 + a) \bmod 2$

(any function can be implemented using these gates)

MPC Construction Idea, III

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, IV

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, V

- 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, VI

- 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 $s_i = de/m + dy_i + ex_i + z_i$ share calculation

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

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

$$= ab$$

Constructing Beaver Triples

- Use interaction between parties.
  - Tool : additive homomorphic encryption:
    - it enables:

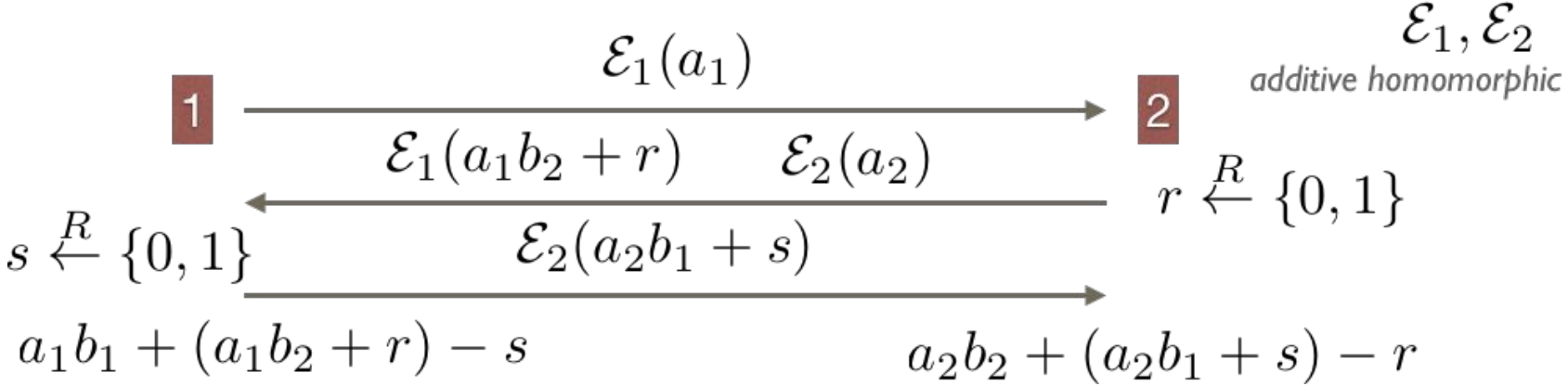
$$\mathcal{E}(x) \cdot \mathcal{E}(y) = \mathcal{E}(x + y \bmod 2) \qquad a, b, \mathcal{E}(x) \Rightarrow \mathcal{E}(ax + b)$$

Constructing Beaver Triples, II

$$(\sum_{i=1}^{2} a_i)(\sum_{i=1}^{2} b_i) = a_1b_1 + a_2b_2 + a_1b_2 + a_2b_1$$

case m=2

$$= \underbrace{(a_1b_1 + a_1b_2 + r - s)}_{s_1} + \underbrace{(a_2b_2 + a_2b_1 + s - r)}_{s_2}$$



MPC strengths and weaknesses

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

Workarounds for fairness

- Optimistic fairness (by involving a third party). The protocol is basically not fair, but a third party is guaranteed to be able to engage and amend the execution in case of an abort/deviation.
- Gradual/timed release. Such protocols engage in many rounds where parties gradually come closer to computing the output (“gradual closeness” can be measured in terms of probability of guessing the output or # of computational steps remaining).

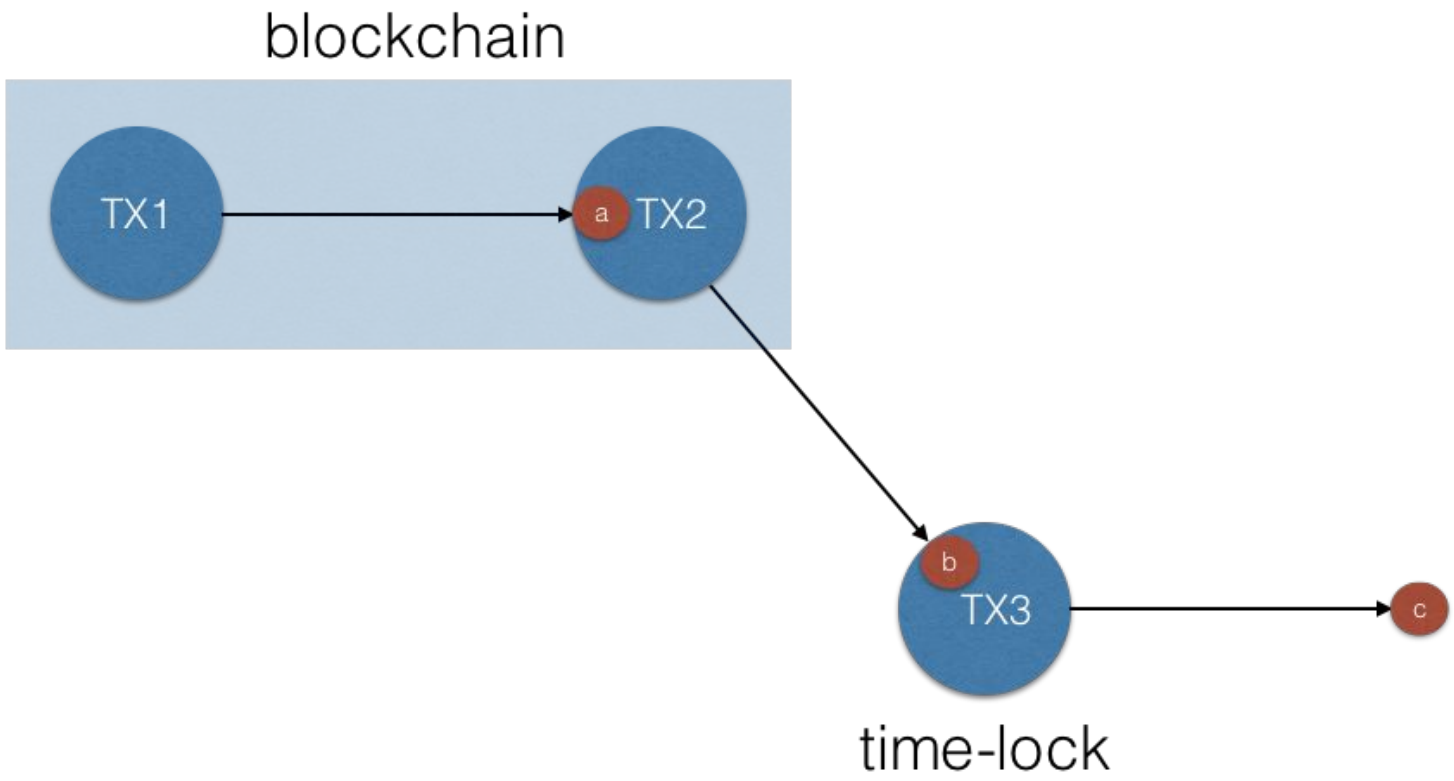
Using a blockchain

- 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

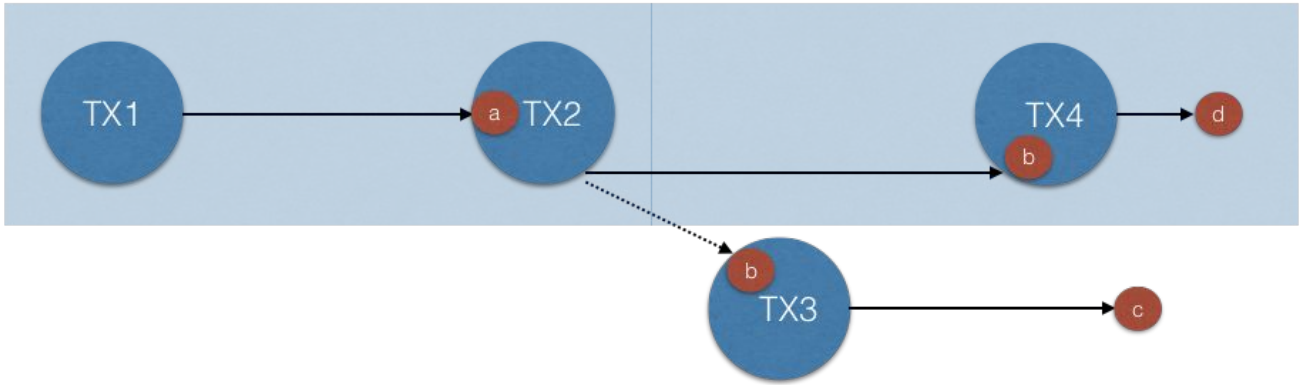
- Time-lock transactions
  - part of transaction data : specifies the earliest time that a transaction can be included in a block.
- Key observation: if a conflicting transaction has already being included in the ledger then the time-lock transaction will be rejected.

Time-lock example



Time-lock example, II

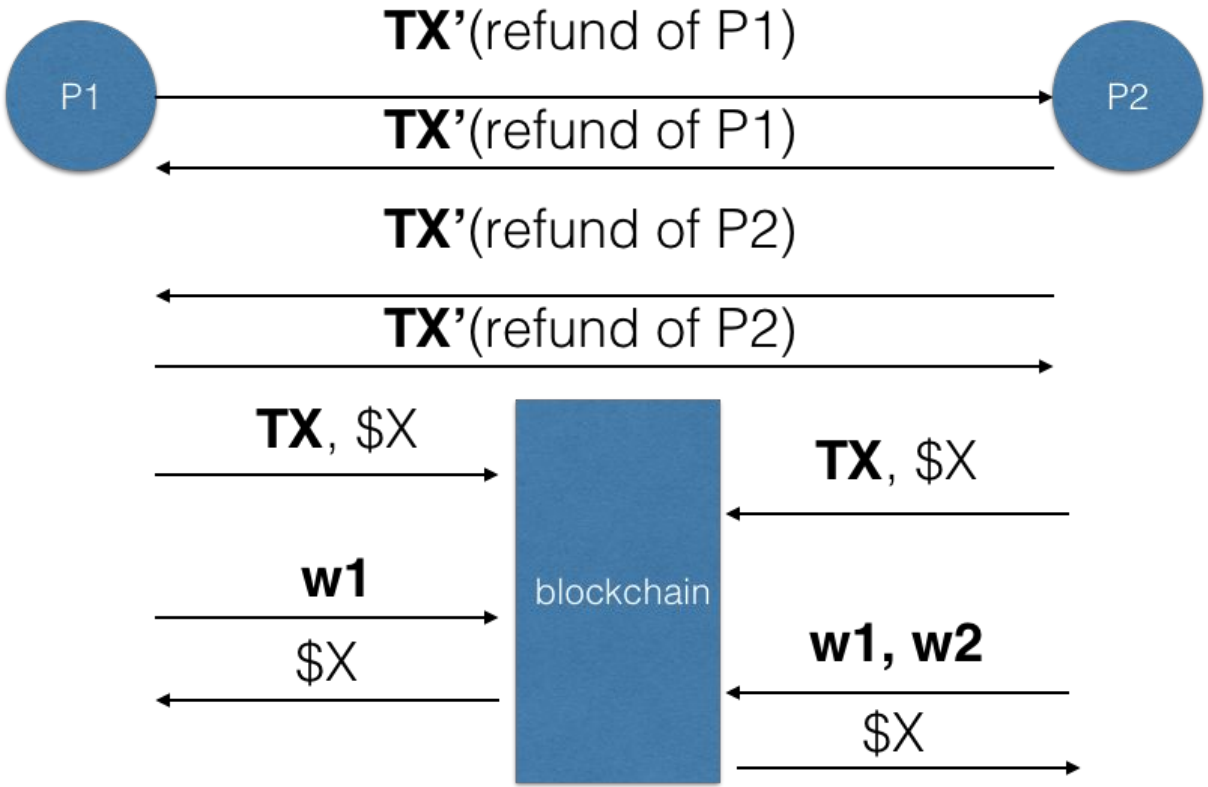
OR



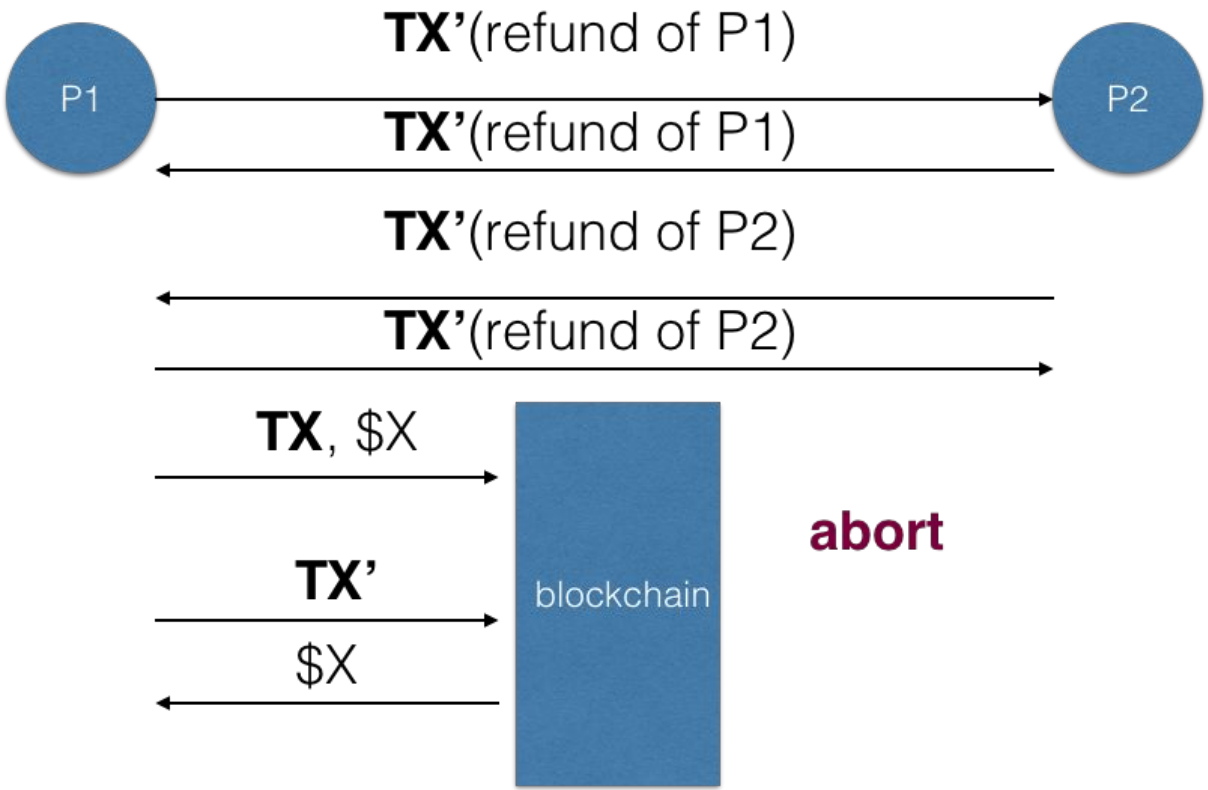
Fair swap of values using time-locks

- P1 holds **w1, h2=H(w2).** P2 holds **w2, h1=H(w1)**
- P1 creates a P2SH transaction **TX** that can be redeemed for $X provided that: (P1 and P2 sign, as 2-out-of-2 multisignature) or (H(**w1**)=**h1** and H(**w2**)=**h2** and P2 signs). P1 also creates a P2PKH **TX'** that spends **TX output with a time-lock in the near future.** P1 sends **TX'** to P2 to sign it. (P2 does not see **TX,** only the transaction ID is needed to refer to it).
- P2 acts in the same way by creating a **TX** that can be redeemed via a 2-out-of-2 multisig or (H(**w1**)=**h1** and P1 signs it)
- P1 by publishing **TX** enables P2 to redeem $X revealing **w1, w2**. P2 by publishing its own **TX** enables P1 to redeem $X revealing **w1**.
- If either party does not reveal their **w** value then the other can claim $X after the time-lock is activated by signing & publishing **TX'**.

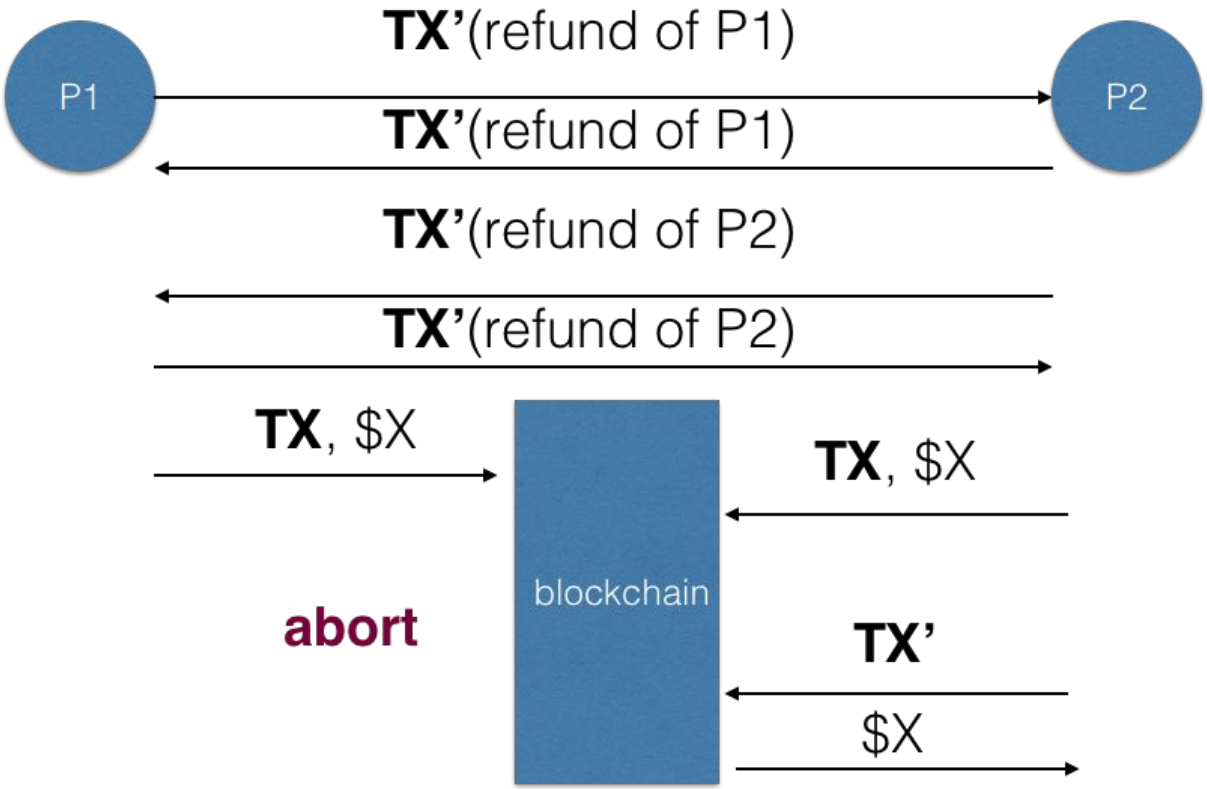
Fair swap of values using time-locs, II



Fair swap of values using time-locks, II



Fair swap of values using time-locks, III



Fair swap of values using time-locks, IV

- If P1's **TX** can be redeemed by "H(**w2**)=**h2** and P2 signs it" then P2 can rush to obtain payment of $X by revealing **w2** while hold back from publishing its own **TX** transaction.
  (note that there is no way to ensure that the respective **TX** transactions will appear concurrently in the blockchain).
- If a multisig is not used for the refunds, then a player may rush to obtain its refund after submitting its value thus invalidating the **TX** payment of the other player.

Fair Computation

- Have the two parties use MPC to compute a secret sharing of the output of the computation, i.e.,
  w1 + w2 = MPC_output
- Subsequently have parties do a fair swap of values, to obtain the MPC_output.
  - If a party aborts, then 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 **w1**,**w2**, …, **wN**. (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 **w1, w2,…,w*N-i***. (The *N-1* parties also prepare these “ladder” **TX** transactions).
- Redeeming follows the sequence P1, P2, …, PN.

N-party ladder construction, II

- P1 will redeem $X from P2 for publishing **w1.**
- P2 will redeem $2X from P3 for publishing **w1,w2.**
- **…**
- PN-1 will redeem $(N-1)X from PN for publishing **w1,w2, …, wN-1.**
- PN will redeem $X from each of P1,…,PN-1 for publishing **w1,w2, …, wN.**