
HELLO
my name is
DelBrag

delbrag*:

it’s like “bitvm” with computation off chain

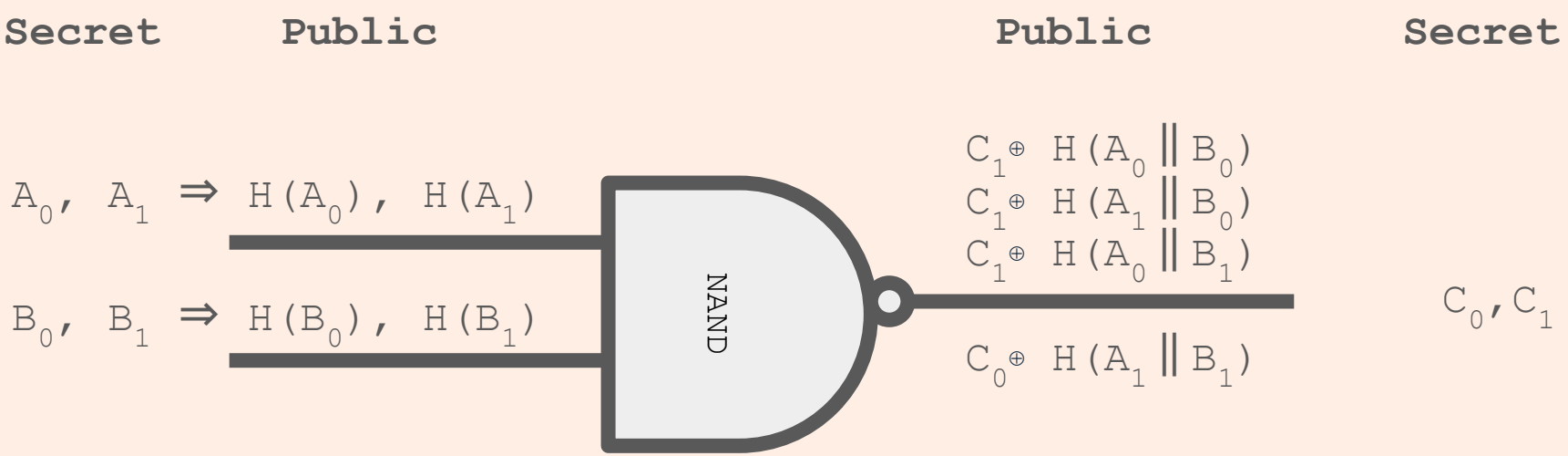
https://rubin.io/public/pdfs/delbrag.pdf

* garbled backwards

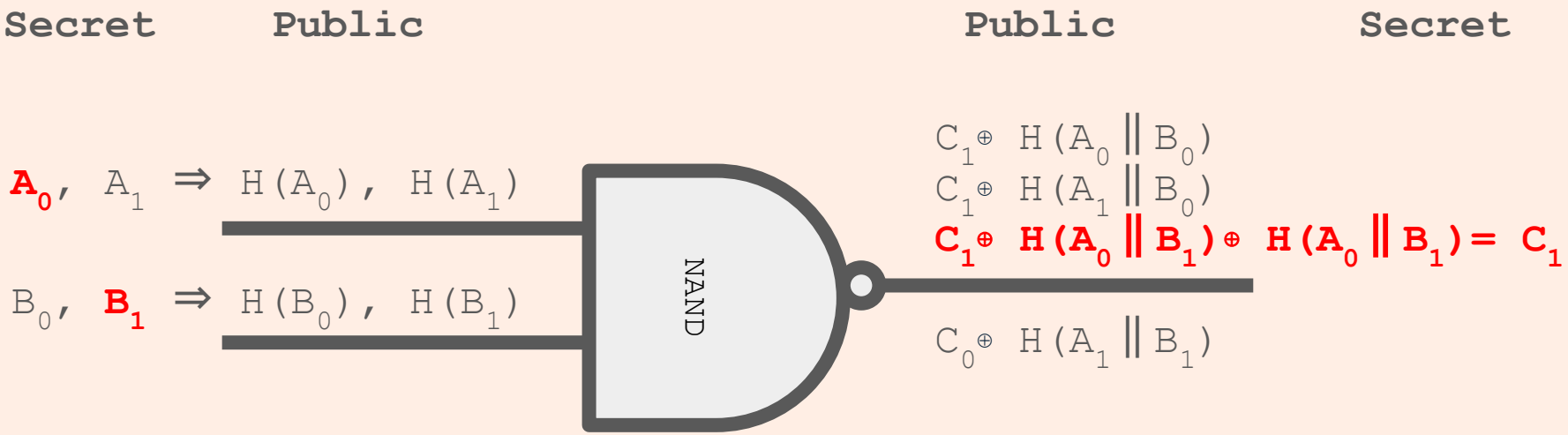
bitvm is cool

extra computation on chain is not

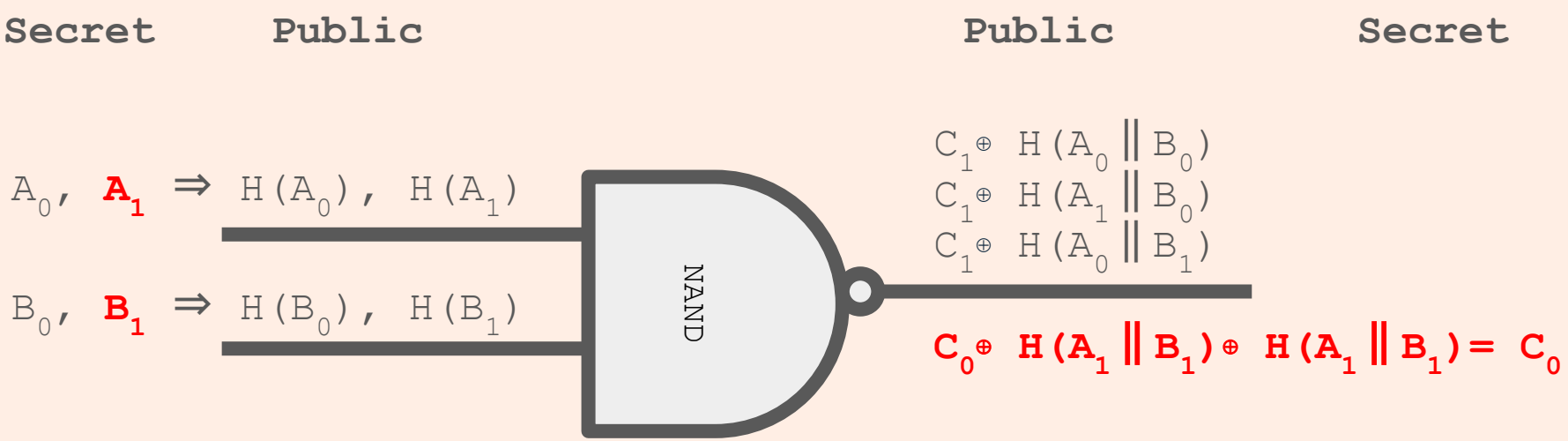
How to (half)-Garble Your Circuit



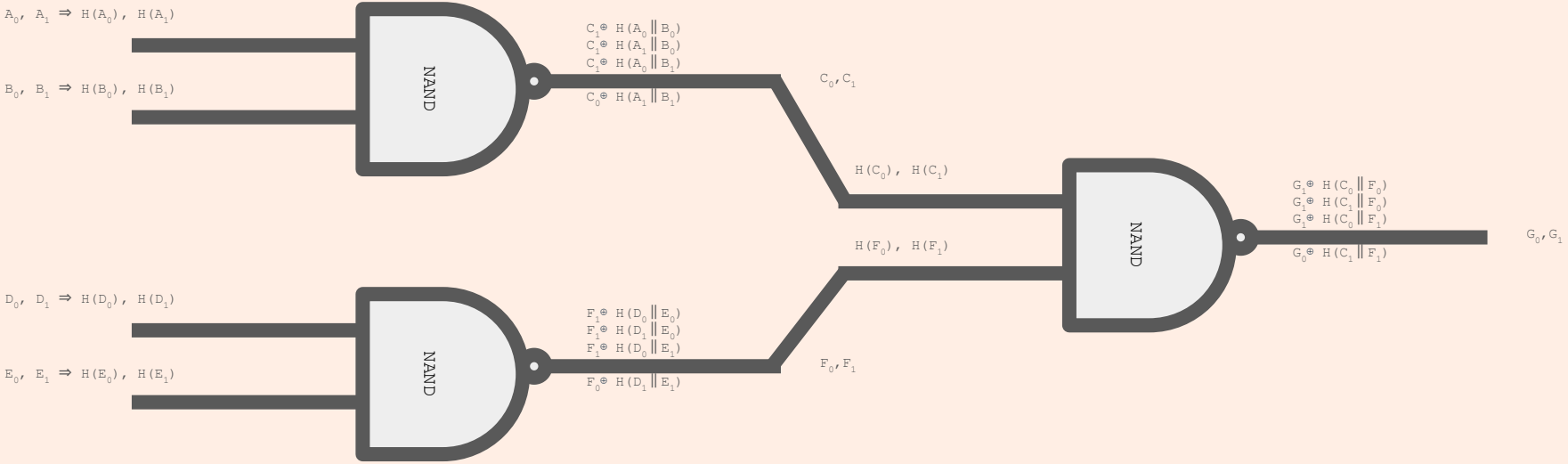
How to (half)-Garble Your Circuit



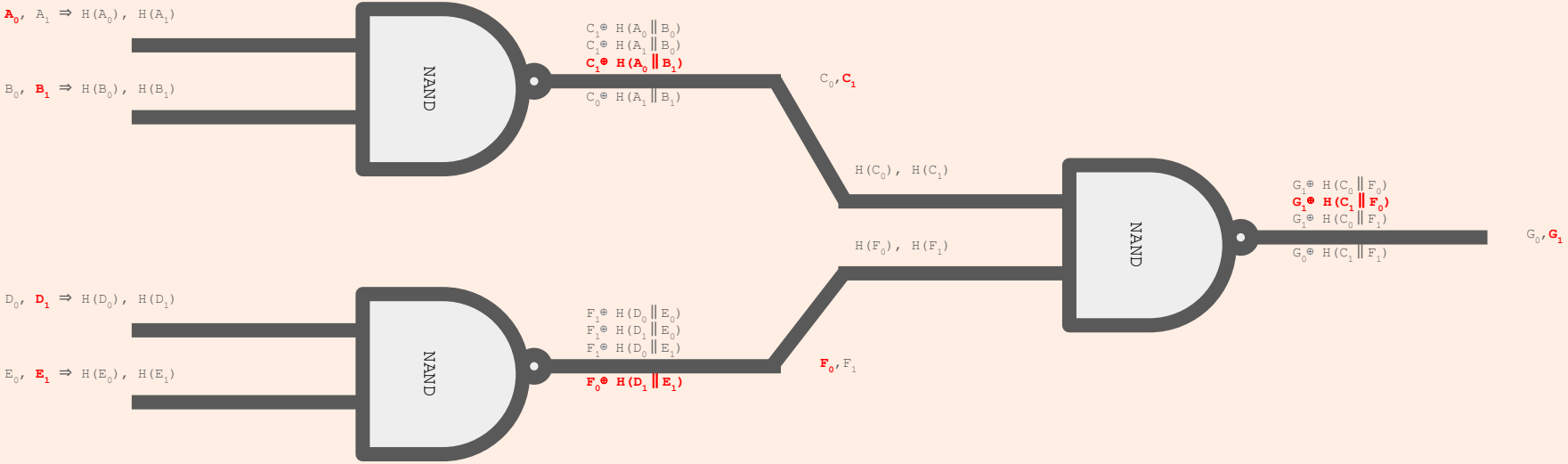
How to (half)-Garble Your Circuit



How to (half)-Garble Your Circuit



How to (half)-Garble Your Circuit



CONSISTENCY CHECK

We need a proof that the circuit is made correctly

- Cut N’ Choose
  - Inefficient
- ZKP
  - Might be slow? Need numbers!
- Correct-by-Construction?

Grug-CONSISTENCY CHECK

Optimistic in script row-checks:

e.g. show that given

H(A_0) and H(B_1)

H(C_1 xor H(A_0 || B_1)) == PRECOMP(H(C_1))

use H=sha256, blake

Reduce to 128-bit for only 2nd preimage resistance for perf

Requires a few minutes of hashing, can be during download, but is secure.

N.b. ensure length checks/fixed length hashes

OP_CAT + OP_XOR would make much more efficient!

(DRAFT) Bob Garbles CONSISTENCY CHECK

Alice creates a homomorphic key K, and sends Bob an encrypted seed S.

Bob creates a gate of i j k, encrypted:

$A_0 = H(S \,||\, i \,||\, 0)$, $A_1 = H(S \,||\, i \,||\, 1)$

$B_0 = H(S \,||\, j \,||\, 0)$, $B_1 = H(S \,||\, j \,||\, 1)$

$C_0 = H(S \,||\, k \,||\, 0)$, $C_1 = H(S \,||\, k \,||\, 1)$

rows: $\mathbf{C_1 \oplus H(A_0 \| B_0)}$, $\mathbf{C_1 \oplus H(A_0 \| B_1)}$, $\mathbf{C_1 \oplus H(A_1 \| B_0)}$, $\mathbf{C_0 \oplus H(A_1 \| B_1)}$

Alice decrypts, reconstructs, checks equality, sends dec key

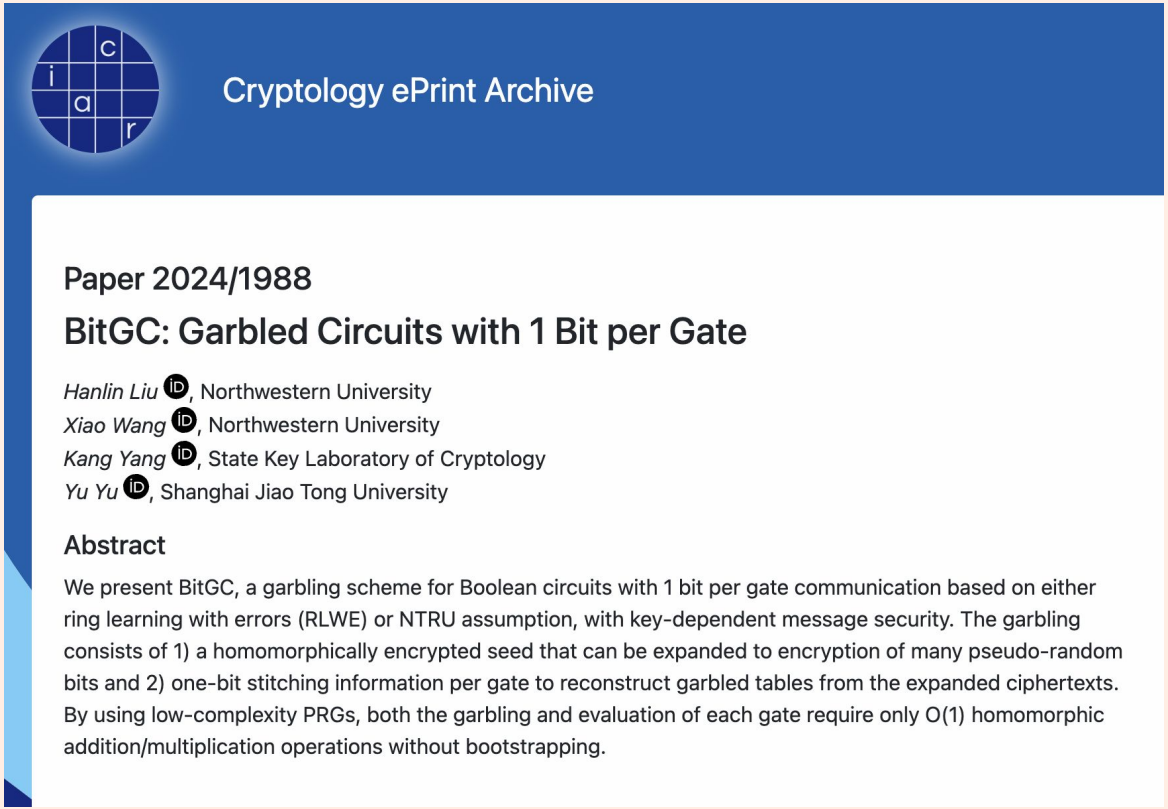
Bob decrypts.

Cost-Per-Gate?

Input: 32 Bytes * 4 + 32 Bytes * 2
Output: 4 * 32 Bytes

Proof overhead: ?

Cost-Per-Gate?



Cryptology ePrint Archive

Paper 2024/1988

BitGC: Garbled Circuits with 1 Bit per Gate

Hanlin Liu, Northwestern University
Xiao Wang, Northwestern University
Kang Yang, State Key Laboratory of Cryptology
Yu Yu, Shanghai Jiao Tong University

Abstract

We present BitGC, a garbling scheme for Boolean circuits with 1 bit per gate communication based on either ring learning with errors (RLWE) or NTRU assumption, with key-dependent message security. The garbling consists of 1) a homomorphically encrypted seed that can be expanded to encryption of many pseudo-random bits and 2) one-bit stitching information per gate to reconstruct garbled tables from the expanded ciphertexts. By using low-complexity PRGs, both the garbling and evaluation of each gate require only O(1) homomorphic addition/multiplication operations without bootstrapping.

Why Half-Garbling?

Traditional “Garbling” also involves exchanging inputs…

We don’t care about that here.

Summary So Far:

- We have built a way for Alice to make a function for Bob:

  $\boldsymbol{\varphi}(\mathbf{X}) \Rightarrow \mathbf{Y}$::CommitReveal[0,1]

- After Alice reveals $\mathbf{X}$, Bob can compute $\boldsymbol{\varphi}$ learning $\mathbf{Y}_0$ or $\mathbf{Y}_1$

Connecting Back to Bitcoin

Output **δ** = SEND 1 BTC to:

```
Tr(MuSig2(Alice, Bob),{
    <H(Y_0)> SHA256 EQUALVERIFY <Bob> CHECKSIG,

    <N> CSV <Alice> CHECKSIG
})
```

What do we have now?

Alice creates $\varphi(X) \Rightarrow Y$ and sends to Bob

Alice creates output δ and sends to Bob (incl Descriptor)

If Alice publishes data Q for X, if $\varphi(Q) = 0$, Bob can Punish

At any time…

Alice & Bob can cooperate

After a delay, Alice refunded

THE CIRCUIT EVALUATION IS ALL OFF-CHAIN

VARIANT:

Output **δ** = SEND 1 BTC to:

```
Tr(MuSig2(Alice, Bob),{
    <H(Y_0)> SHA256 EQUALVERIFY <Bob> CHECKSIG,

    <N> CSV <Alice> CHECKSIGVERIFY <Bob> CHECKSIG
})
```

Bob presigns Alice's refund tx

VARIANT:

```
Output δ = SEND 1 BTC to:

Tr(MuSig2(Alice, Bob),{
    <H(Y_0)> SHA256 EQUALVERIFY <Bob> CHECKSIGVERIFY
    <Alice> CHECKSIG,

    <N> CSV <Alice> CHECKSIG
})
```

Alice Presigns Bob's punishment

VARIANT: Grug-Consistency

Output δ = SEND 1 BTC to: Tr(MuSig2(Alice, Bob),{

<H(Y_0)> SHA256 EQUALVERIFY <Bob> CHECKSIGVERIFY

<Bob> CHECSKIGVERIFY ∀R ∈ circuit truth table rows, a leaf with:

Data independent prefix

VOP_DUP VOP_TOALT VOP_BLAKE VOP_TOALT \\ get $R{\rightarrow}A_x$, BLAKE($R{\rightarrow}A_x$) on AltStack

VOP_DUP VOP_TOALT VOP_BLAKE VOP_TOALT \\ get $R{\rightarrow}B_y$, BLAKE($R{\rightarrow}B_y$) on AltStack

VOP_ALT_ABCD_TO_ALT_BD_REG_AC \\ alt: BLAKE($R{\rightarrow}A_x$)BLAKE($R{\rightarrow}B_y$), reg:$R{\rightarrow}A_x$ $R{\rightarrow}B_y$

VOP_CAT VOP_BLAKE VOP_TOALT \\ alt: BLAKE($R{\rightarrow}A_x$)BLAKE($R{\rightarrow}B_y$)BLAKE($R{\rightarrow}A_x||R{\rightarrow}B_y$)

VOP_DUP VOP_TOALT VOP_BLAKE VOP_TOALT \\ alt: BLAKE($R{\rightarrow}A_x$)BLAKE($R{\rightarrow}B_y$)BLAKE($R{\rightarrow}A_x||R{\rightarrow}B_y$) $R{\rightarrow}C_{x\ \text{NAND}\ Y}$ BLAKE($R{\rightarrow}C_{x\ \text{NAND}\ y}$)

VOP_ALT_ABC_TO_ALT_C_REG_AB \\ alt: BLAKE($R{\rightarrow}A_x$)BLAKE($R{\rightarrow}B_y$)BLAKE($R{\rightarrow}C_{x\ \text{NAND}\ y}$) reg: BLAKE($R{\rightarrow}A_x||R{\rightarrow}B_y$) $R{\rightarrow}C_{x\ \text{NAND}\ Y}$

VOP_XOR \\ alt: BLAKE($R{\rightarrow}A_x$)BLAKE($R{\rightarrow}B_y$)BLAKE($R{\rightarrow}C_{x\ \text{NAND}\ y}$) reg: <BLAKE($R{\rightarrow}A_x||R{\rightarrow}B_y$) xor $R{\rightarrow}C_{x\ \text{NAND}\ Y}$ >

<PRECOMP(BLAKE($R{\rightarrow}A_x||R{\rightarrow}B_y$) xor $R{\rightarrow}C_{x\ \text{NAND}\ Y}$)> VOP_EQUALVERIFY // Check the ciphertext is the gate's

VOP_FROMALT <PRECOMP(BLAKE($R{\rightarrow}C_{x\ \text{NAND}\ y}$))> VOP_***NOT***EQUALVERIFY // fraud check the output wire

VOP_FROMALT <PRECOMP(BLAKE($R{\rightarrow}A_x$))> VOP_OTEQUALVERIFY // check the input was correct

VOP_FROMALT <PRECOMP(BLAKE($R{\rightarrow}B_y$))> VOP_NOTEQUALVERIFY // check the input was correct,

<N> CSV <Alice> CHECKSIG

})

Γδ: Γarbleδ δelbraΓ

Let **Γ** be the **G**nostic input.

TR(Musig2(Alice, Bob),{

∀X^i ∈ X…

SHA256 DUP

<H(X^i_0)> EQUAL NOTIF <H(X^i_1)> EQUALVERIFY ENDIF

<Alice> <Bob> CHECKSIG,

<T> CLTV ***[optional <Alice> CHECKSIGVERIFY]*** <Bob> CHECKSIG

})

Γδ: Γarbleδ δelbraΓ

Let **δ** be the **d**iploma input (i.e., a certificate of knowledge)

reminder:

```
TR(Musig2(Alice, Bob),{
    <N> CSV <Alice> CHECKSIG,
    SHA256 <Y_0> EQUALVERIFY <Bob> CHECKSIG
})
```

δ

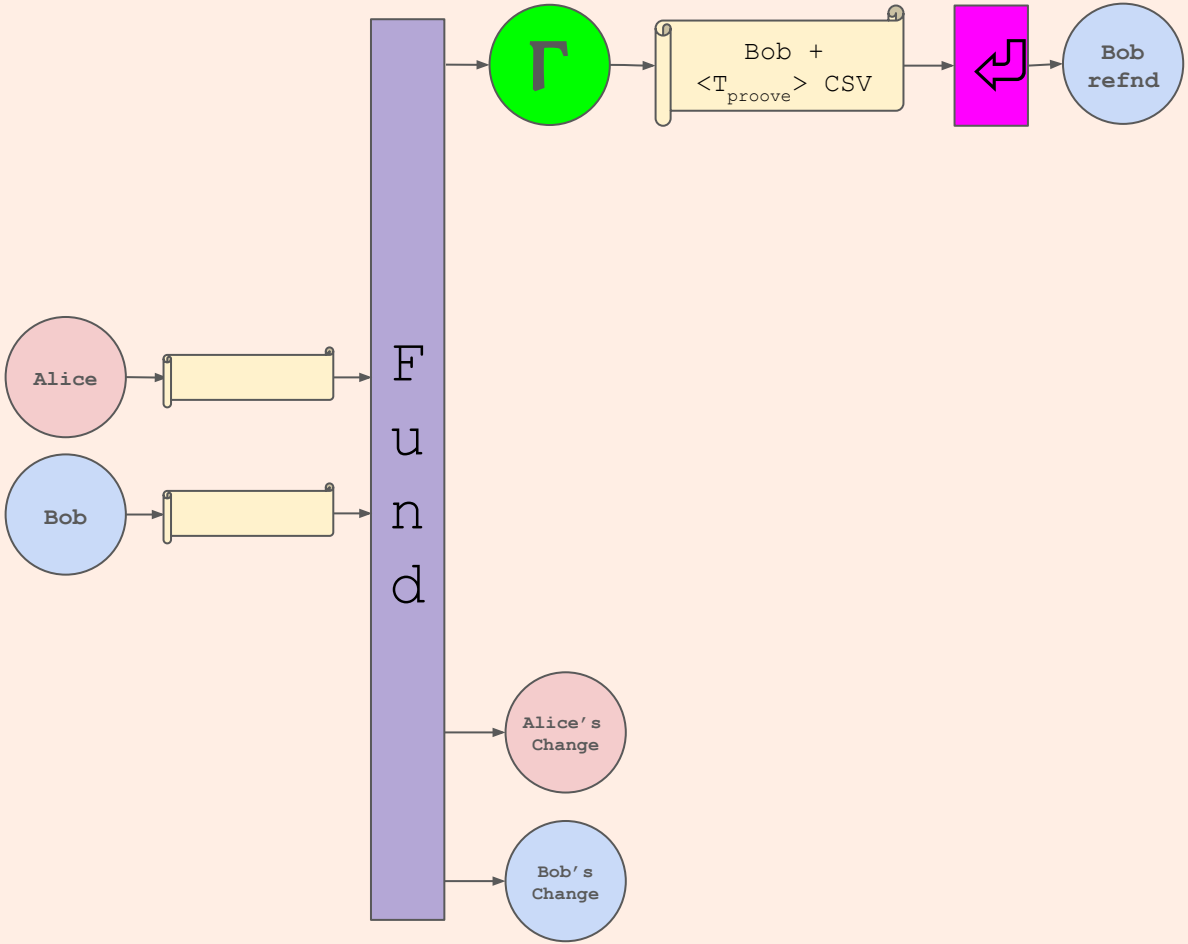
Creating **δ** means that:

input data **x** must have been revealed

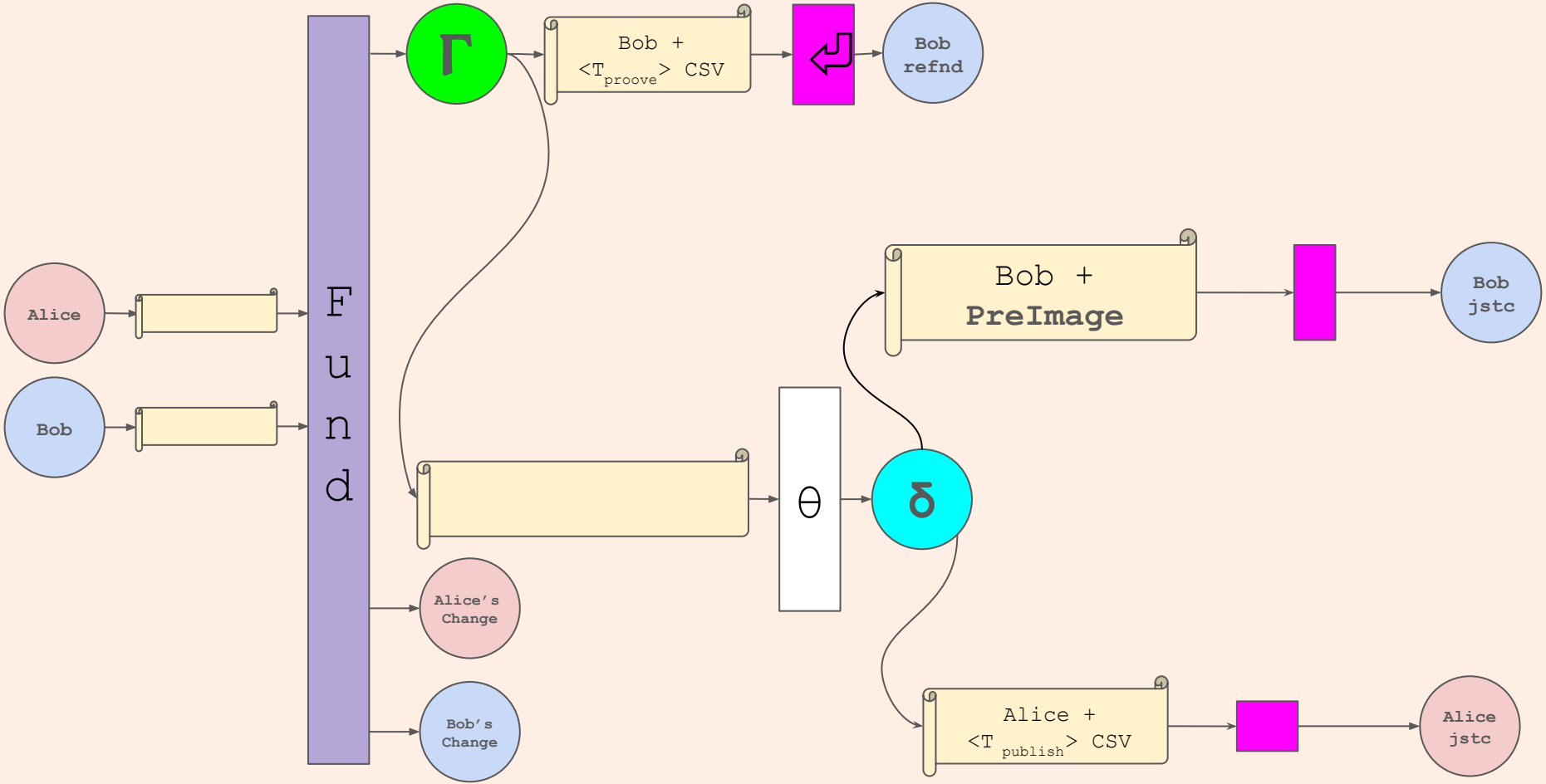
Back to Delbrag-ity

One-Shot Delbrag Protocol

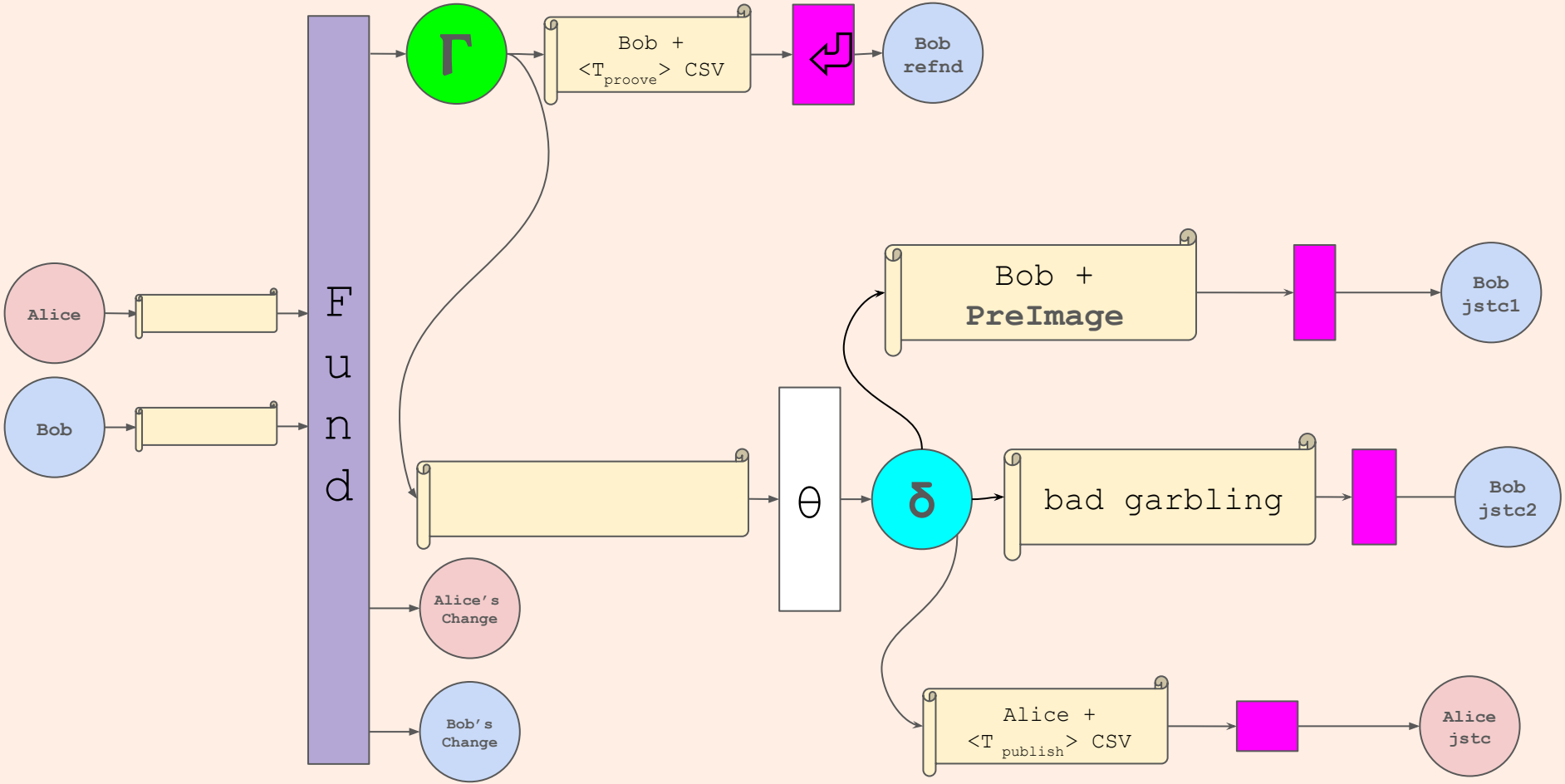
One-Shot Delbrag



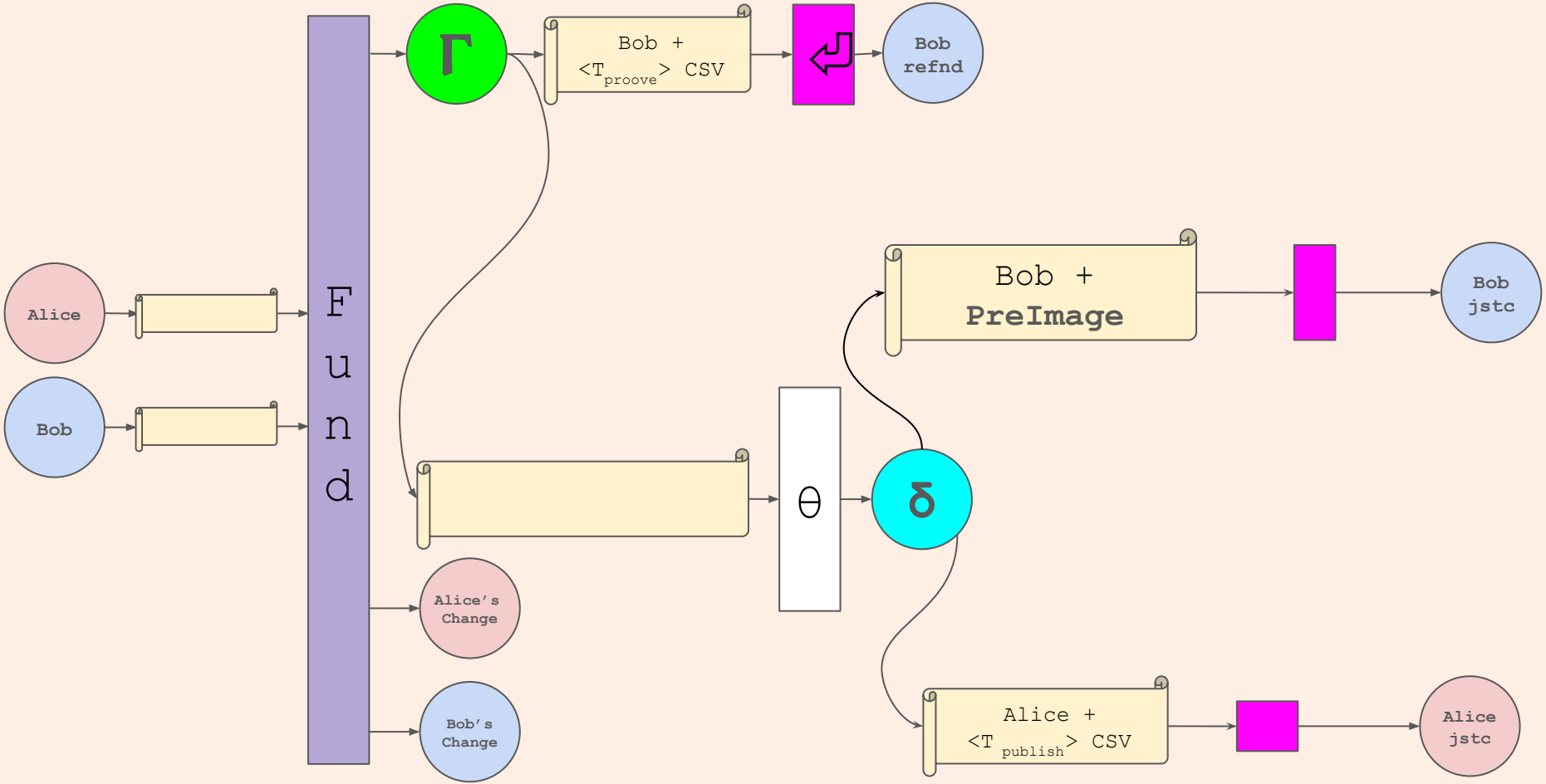
One-Shot Delbrag



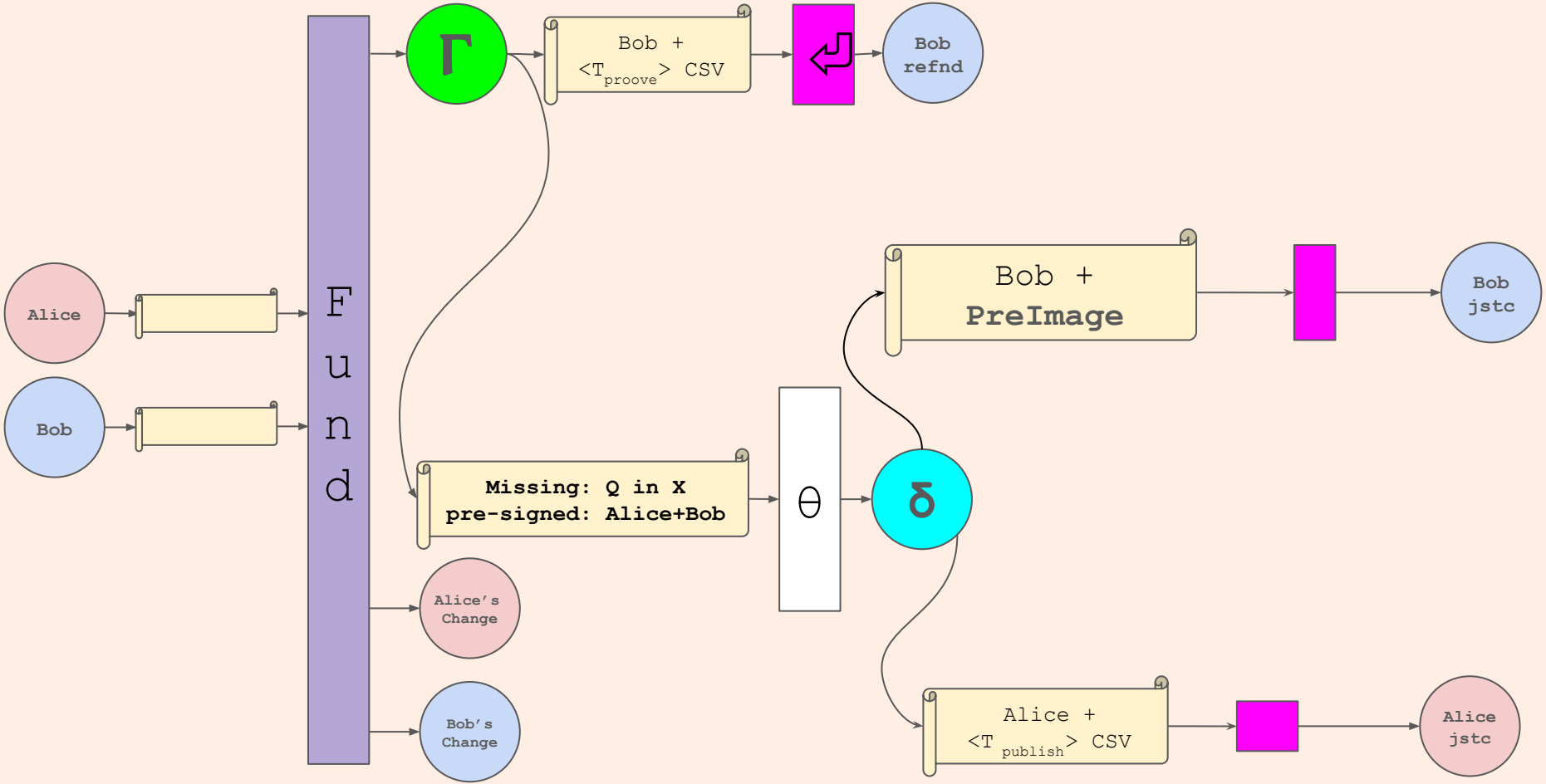
One-Shot Delbrag



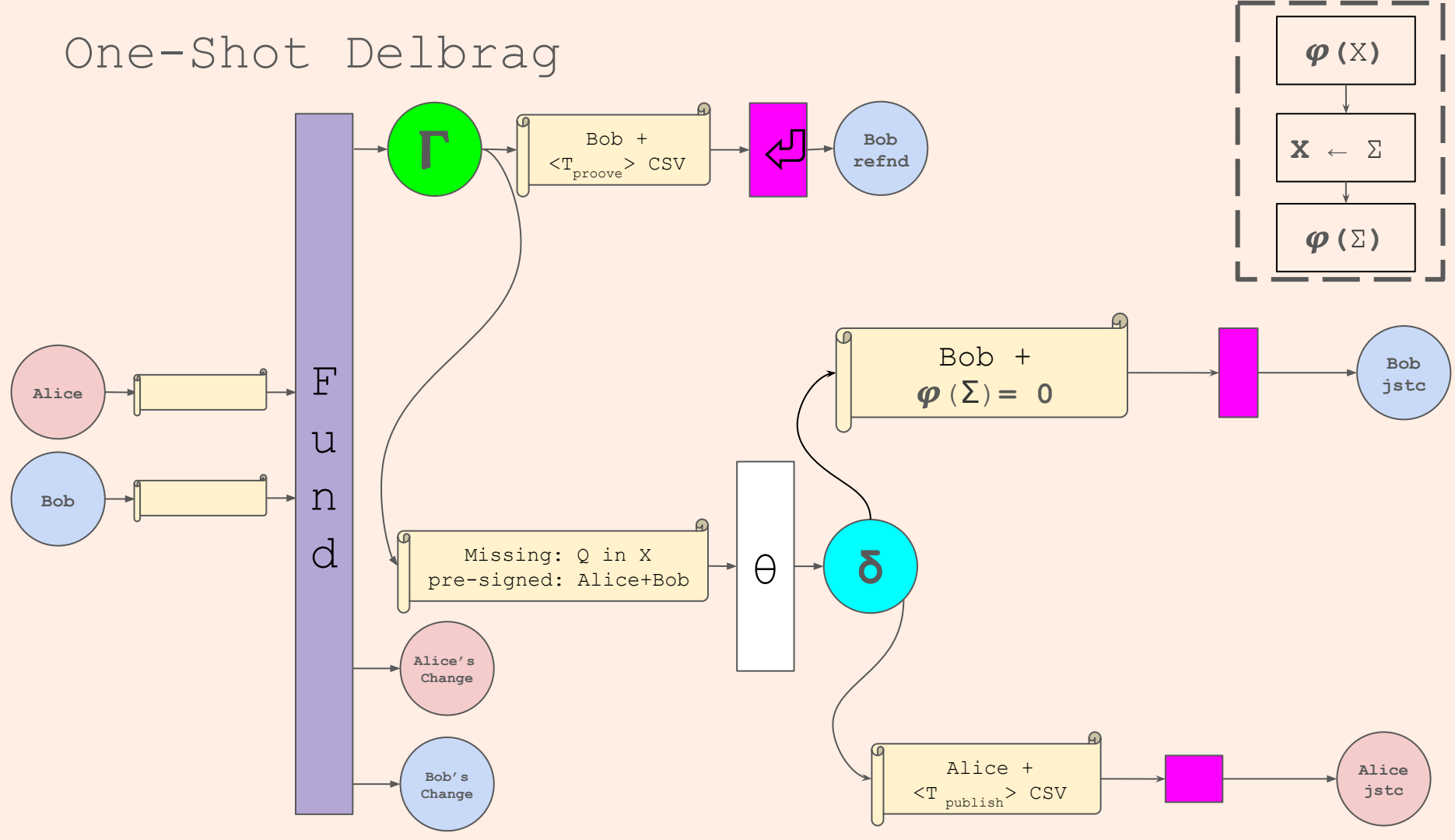
One-Shot Delbrag



One-Shot Delbrag



One-Shot Delbrag

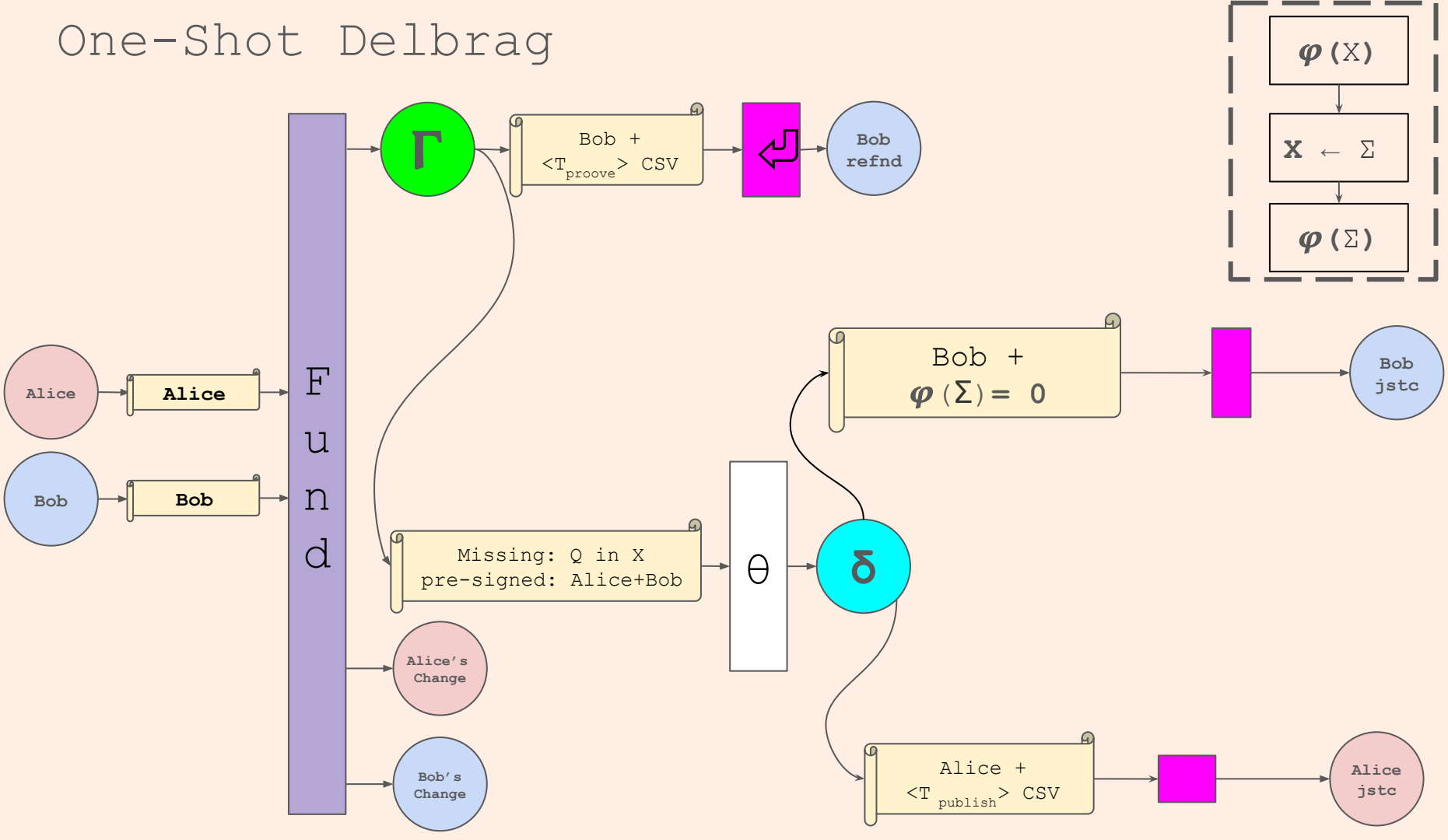


Engineering $\varphi(X)$

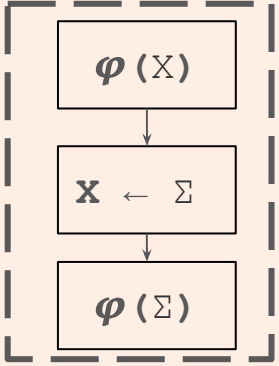
X:

- **Σ**: partially signed **S**pending txn(s)
  - Spending output **δ**
  - Signed by Alice
- Additional “Witness” Data, e.g.:
  - info Bob would need to redeem
  - a ZKP

One-Shot Delbrag



One-Shot Delbrag



Alice

Alice

Bob

Bob

Fund

Γ

Bob +
<T proove> CSV

Bob
refnd

Bob +
φ(Σ) = 0

Bob
jstc

Missing: Q in X
pre-signed: Alice+Bob

Θ

δ

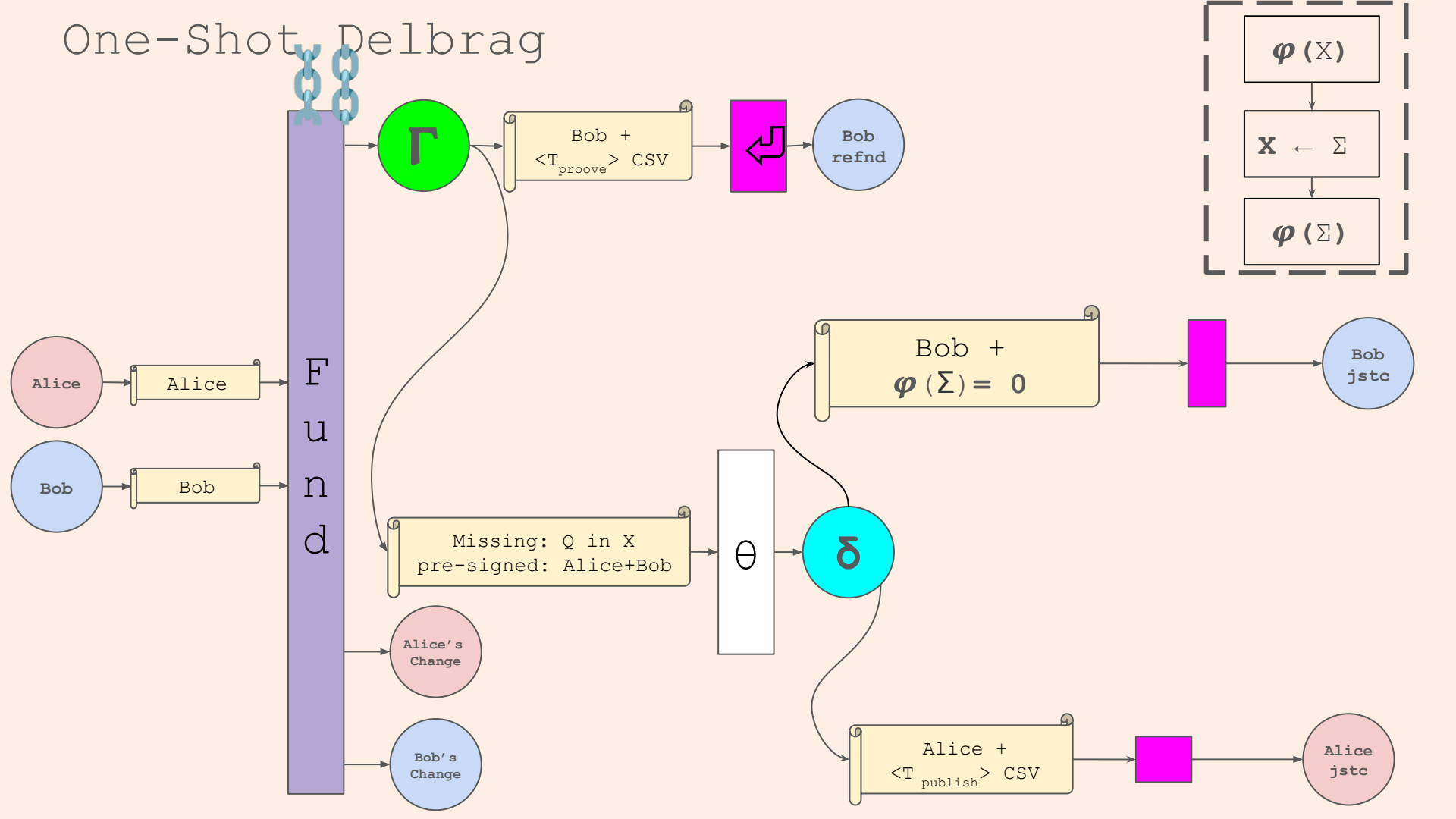
Alice's
Change

Bob's
Change

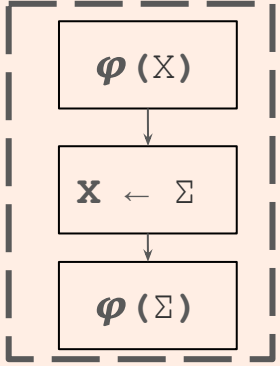
Alice +
<T publish> CSV

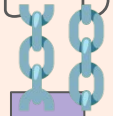
Alice
jstc

One-Shot Delbrag



One-Shot Delbrag





Γ

Bob +
<T_{proove}> CSV

Bob refnd

Alice

Alice

Bob

Bob

Fund

Bob +
φ(Σ) = 0

Bob jstc

Missing: Q in X
pre-signed: Alice+Bob

Θ

δ

PreSig: Alice
Missing: Bob

Σ

Alice dist

Bob dist

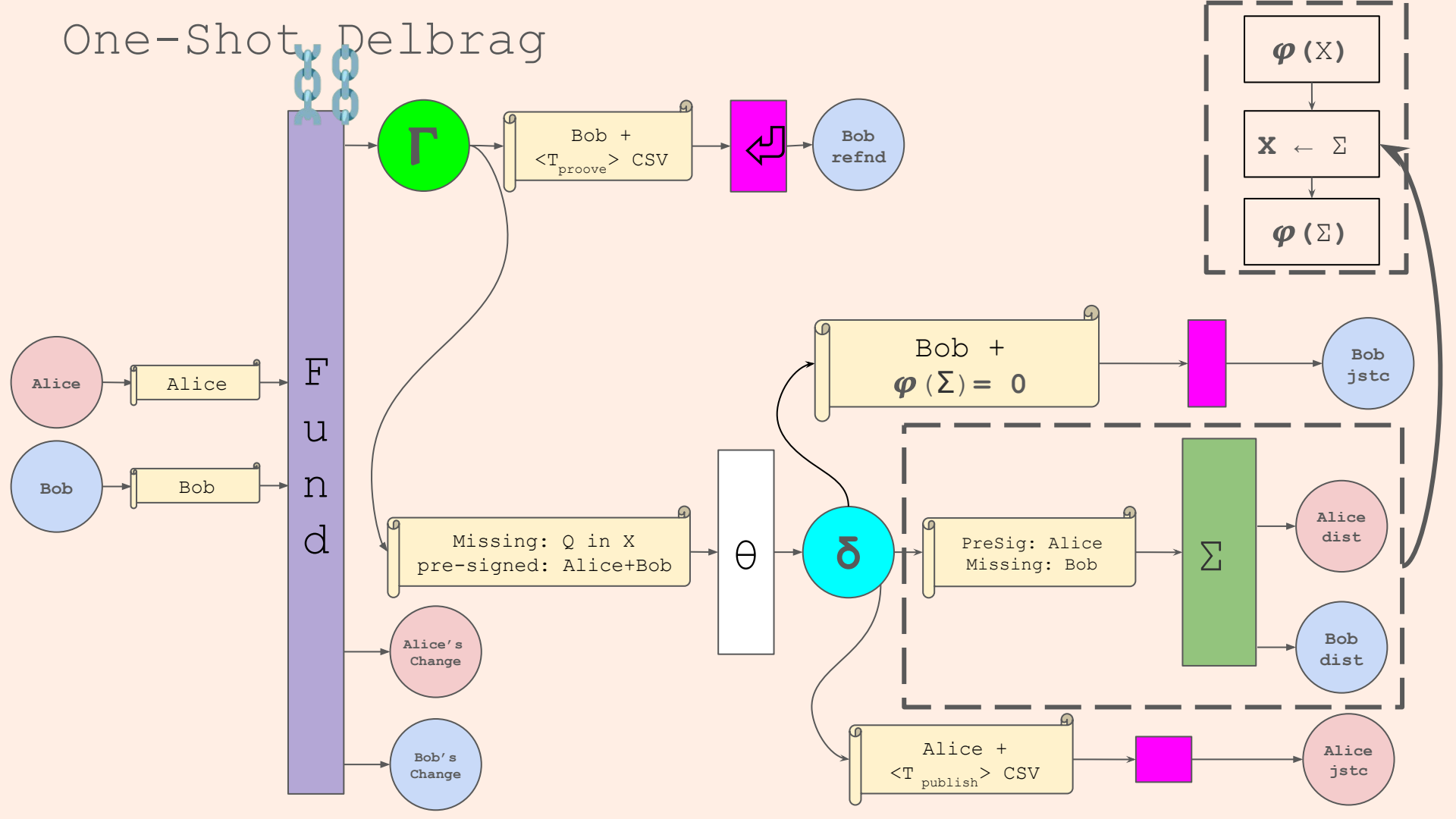
Alice's Change

Bob's Change

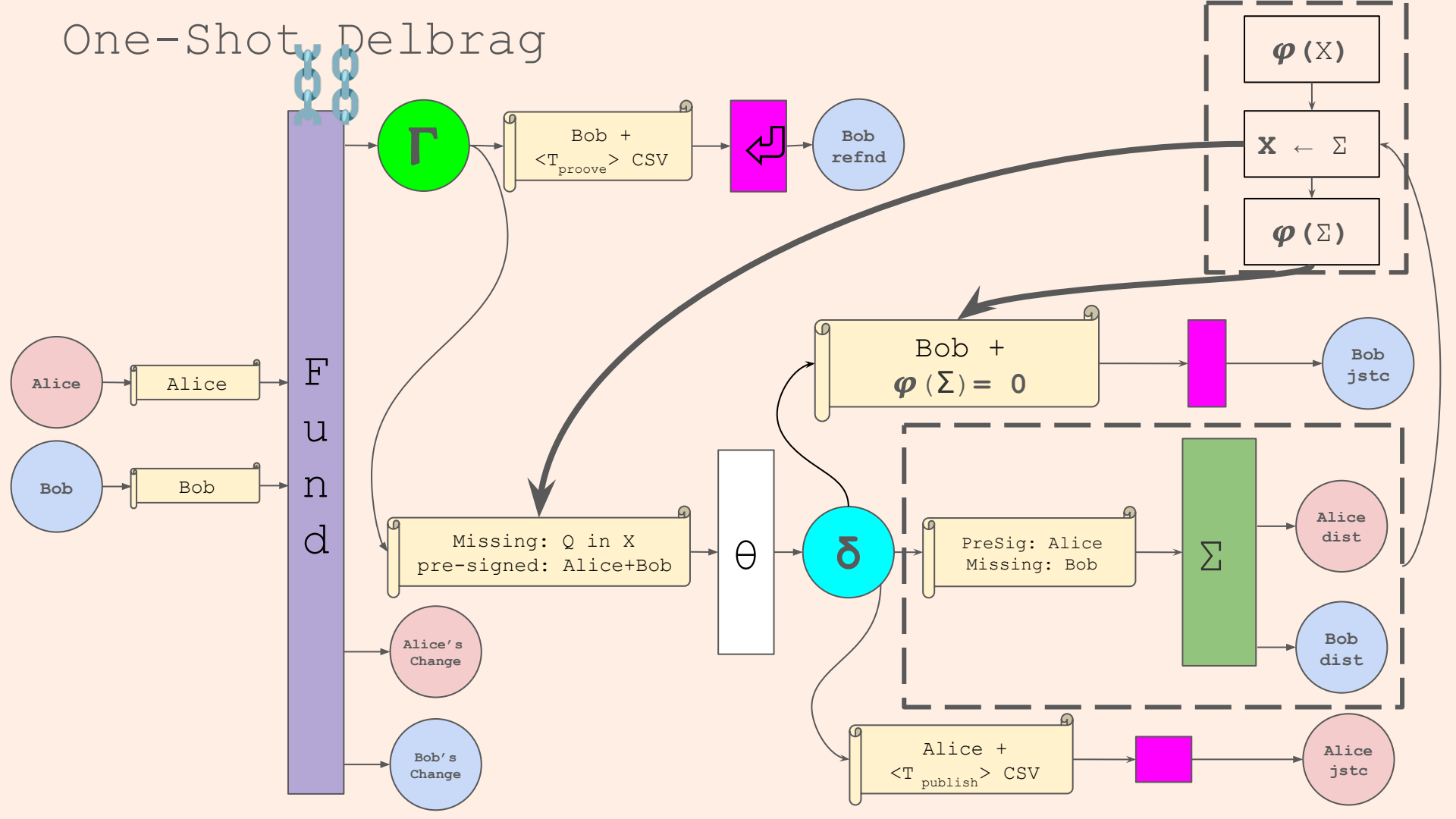
Alice +
<$T_{publish}$> CSV

Alice jstc

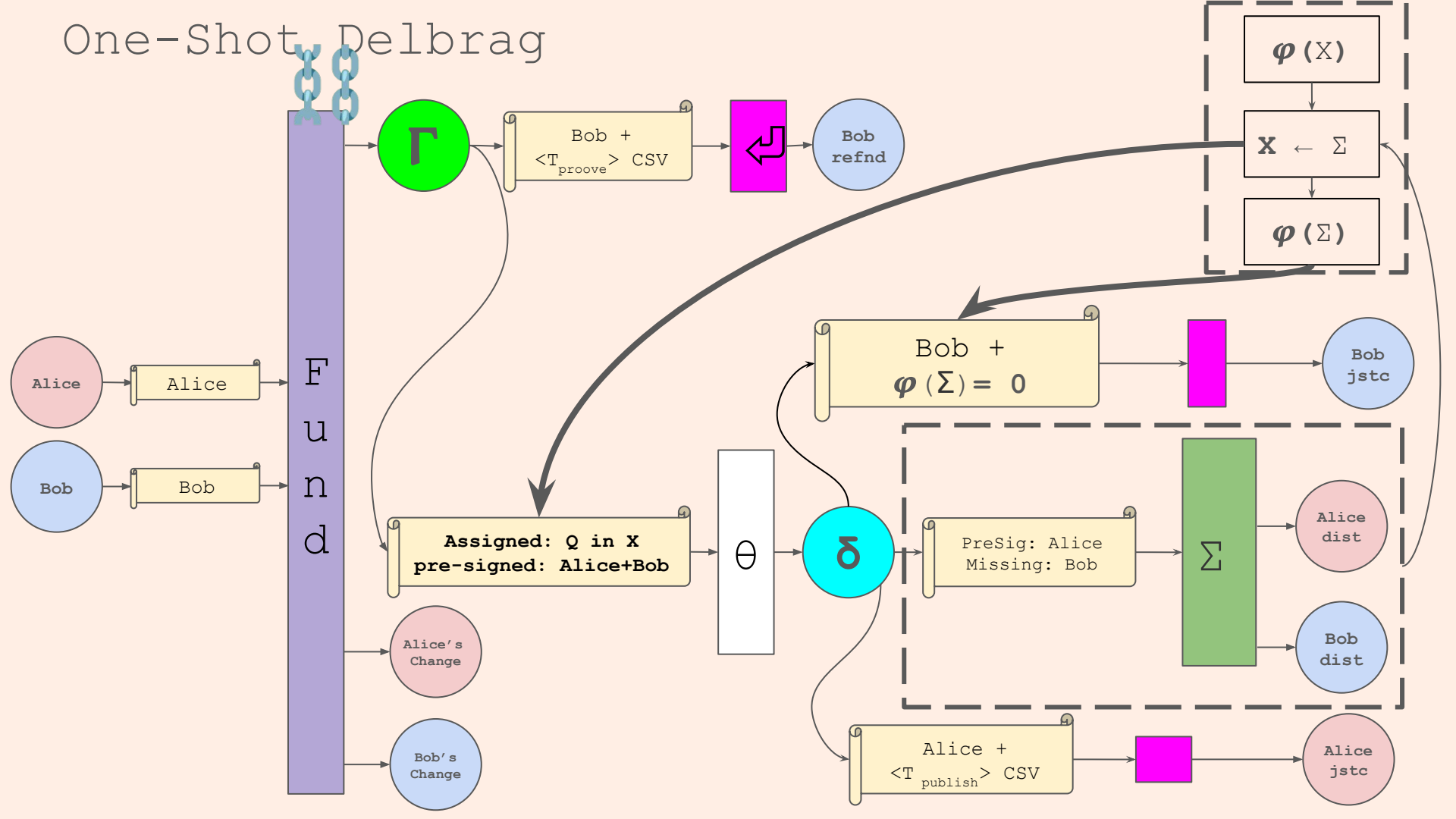
One-Shot Delbrag



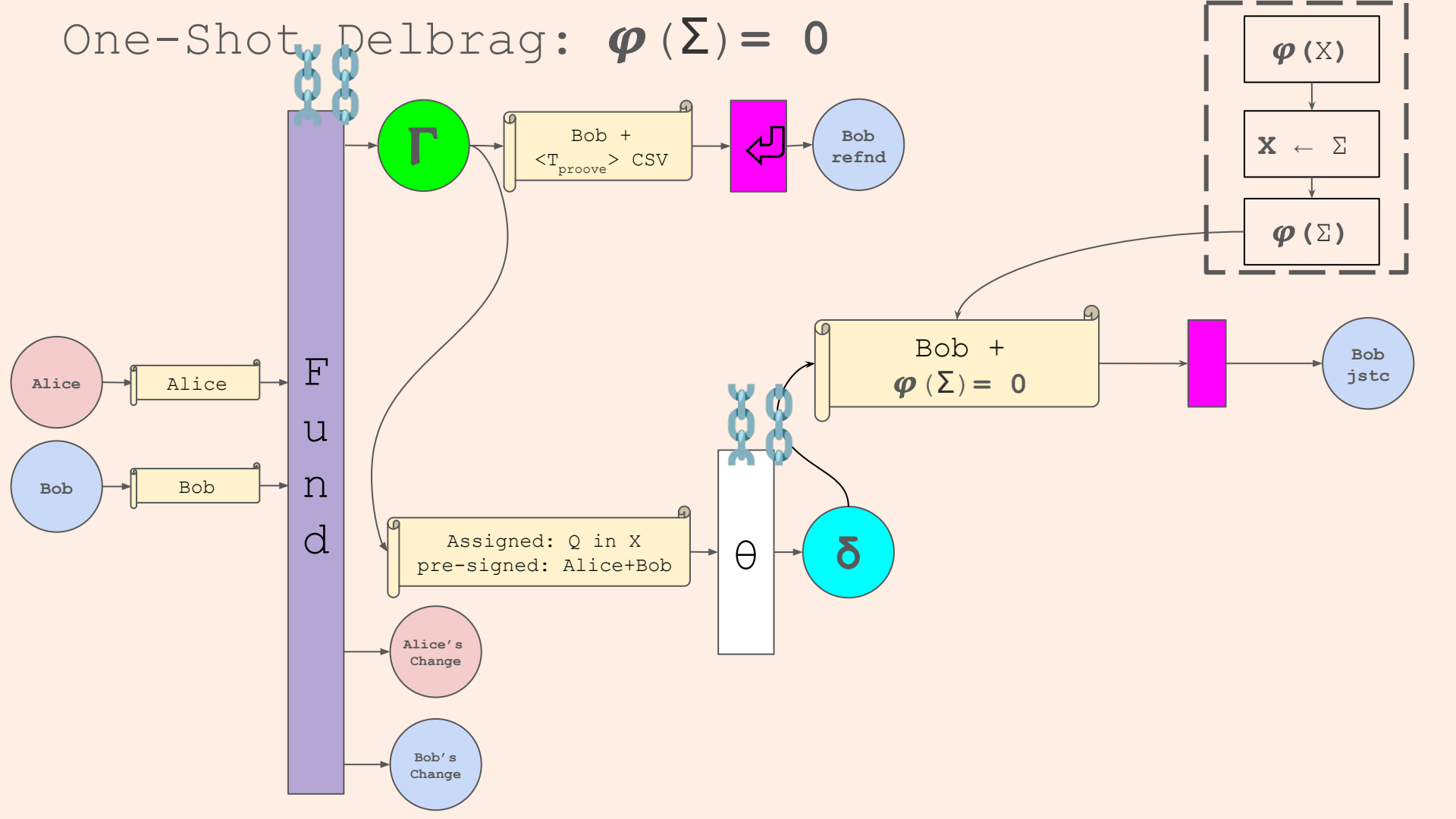
One-Shot Delbrag

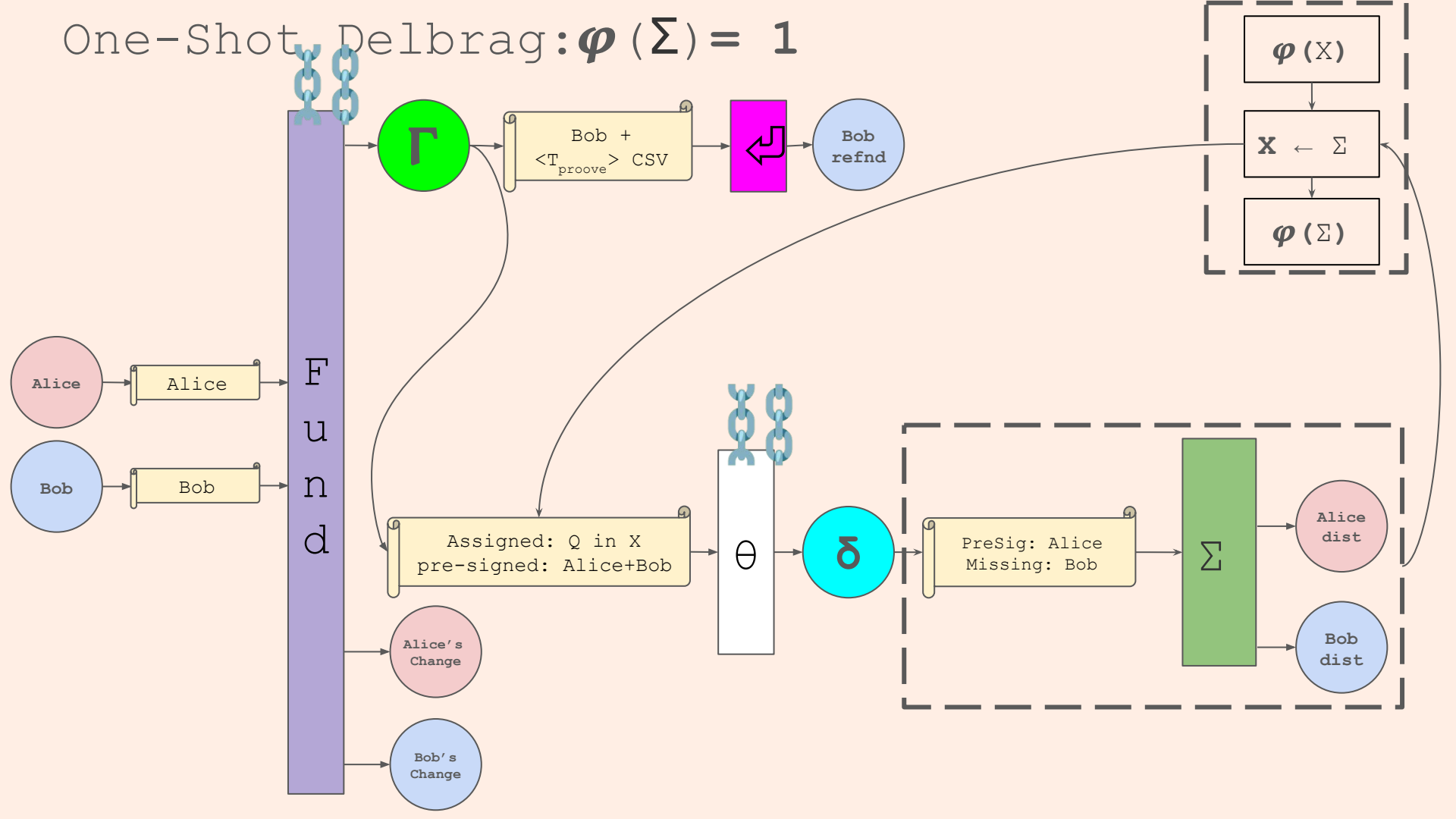


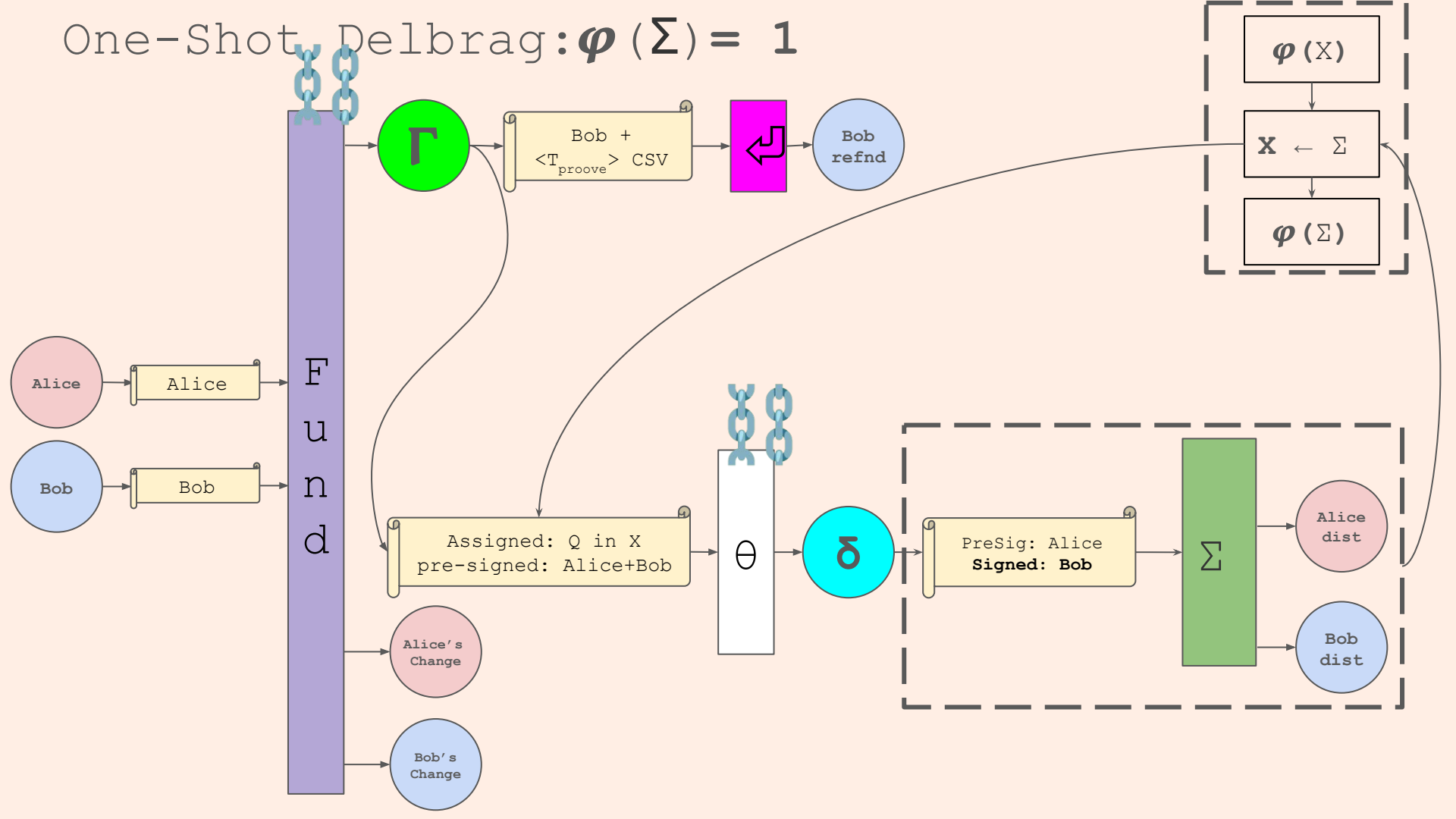
One-Shot Delbrag



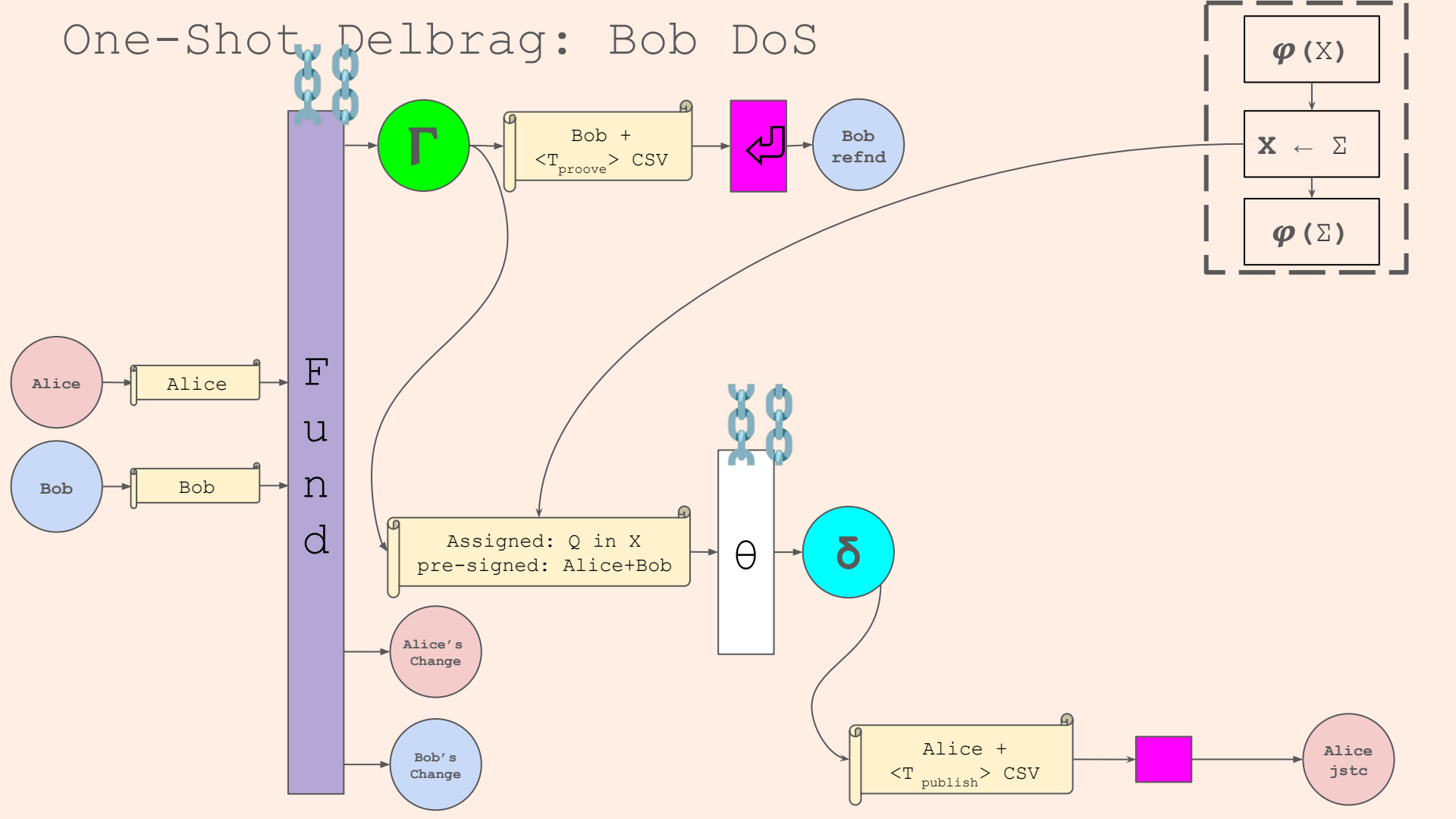
One-Shot Delbrag: $\varphi(\Sigma) = 0$



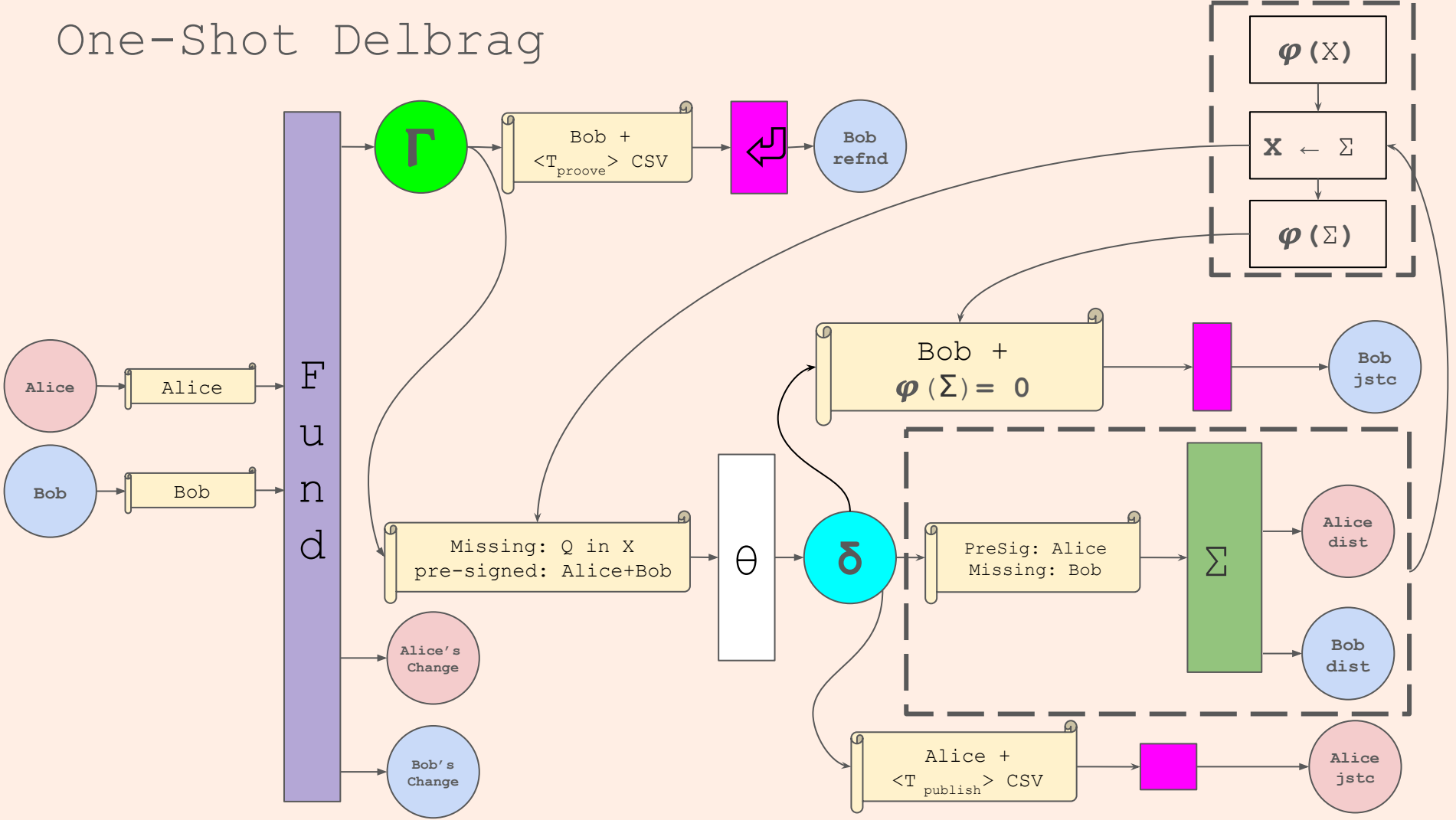

One-Shot Delbrag: φ(Σ) = 1
Alice
Alice
Bob
Bob
Fund
Γ
Bob +
<T proove> CSV
Bob
refnd
Alice's
Change
Bob's
Change
Assigned: Q in X
pre-signed: Alice+Bob
Θ
δ
PreSig: Alice
Missing: Bob
Σ
Alice
dist
Bob
dist
φ(X)
X ← Σ
φ(Σ)


One-Shot Delbrag: φ(Σ) = 1
Γ
Bob + <T_proove> CSV
Bob refnd
φ(X)
X ← Σ
φ(Σ)
Alice
Alice
Bob
Bob
Fund
Assigned: Q in X
pre-signed: Alice+Bob
Θ
δ
PreSig: Alice
Signed: Bob
Σ
Alice dist
Bob dist
Alice's Change
Bob's Change

One-Shot Delbrag: Bob DoS



One-Shot Delbrag

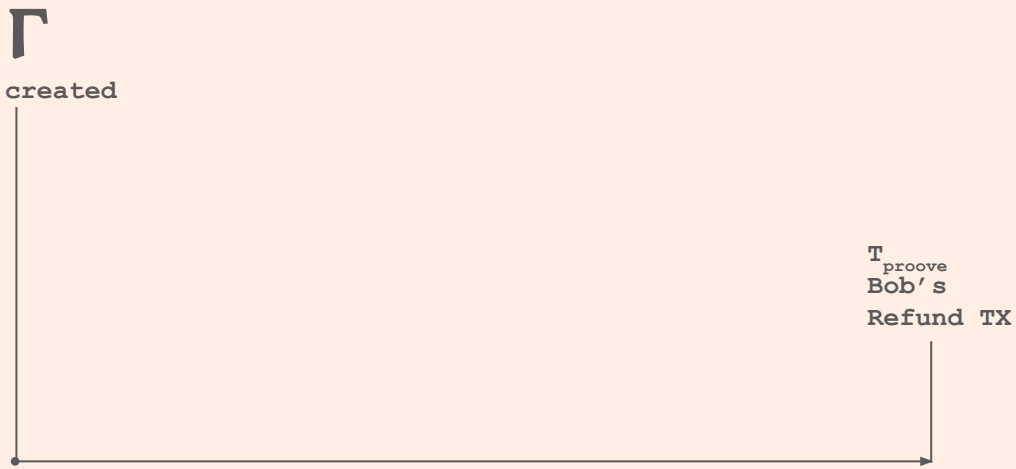


Timing Analysis

Δ = tx inclusion margin

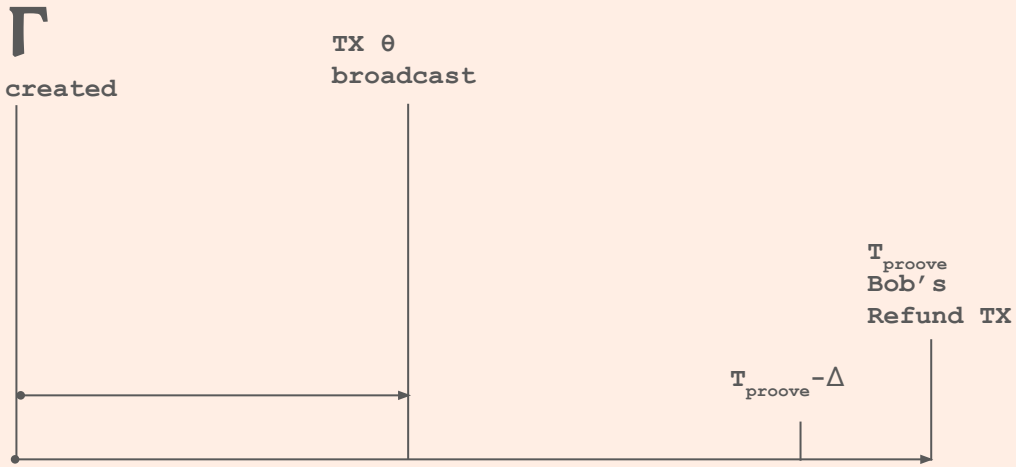
Timing Analysis

Δ = tx inclusion margin



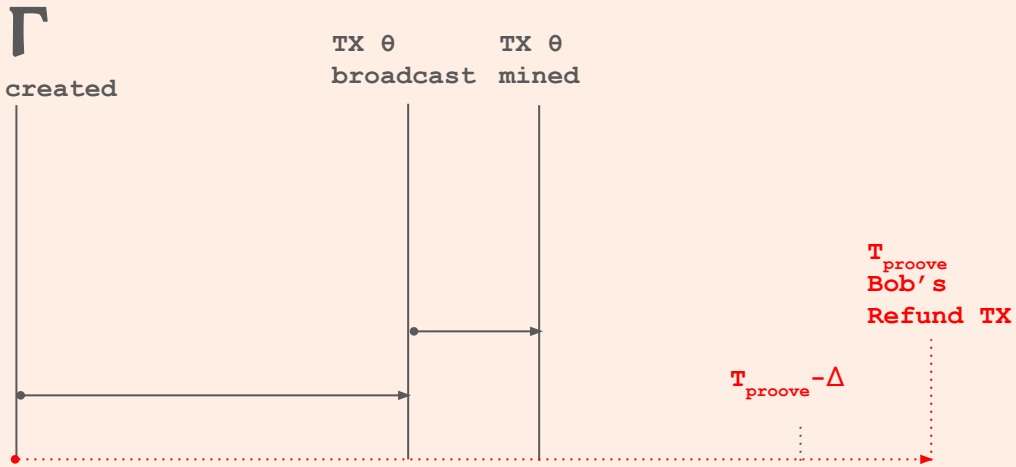
Timing Analysis

Δ = tx inclusion margin



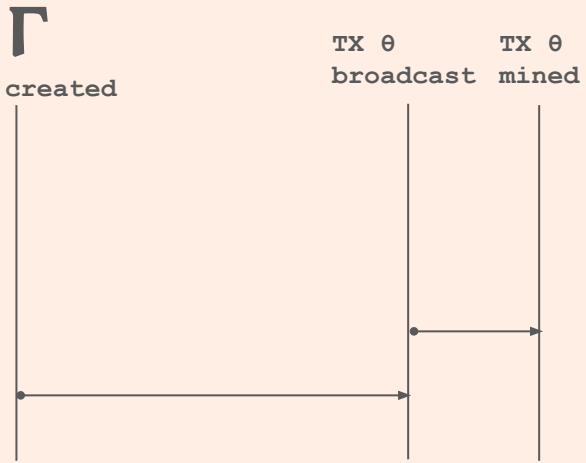
Timing Analysis

Δ = tx inclusion margin



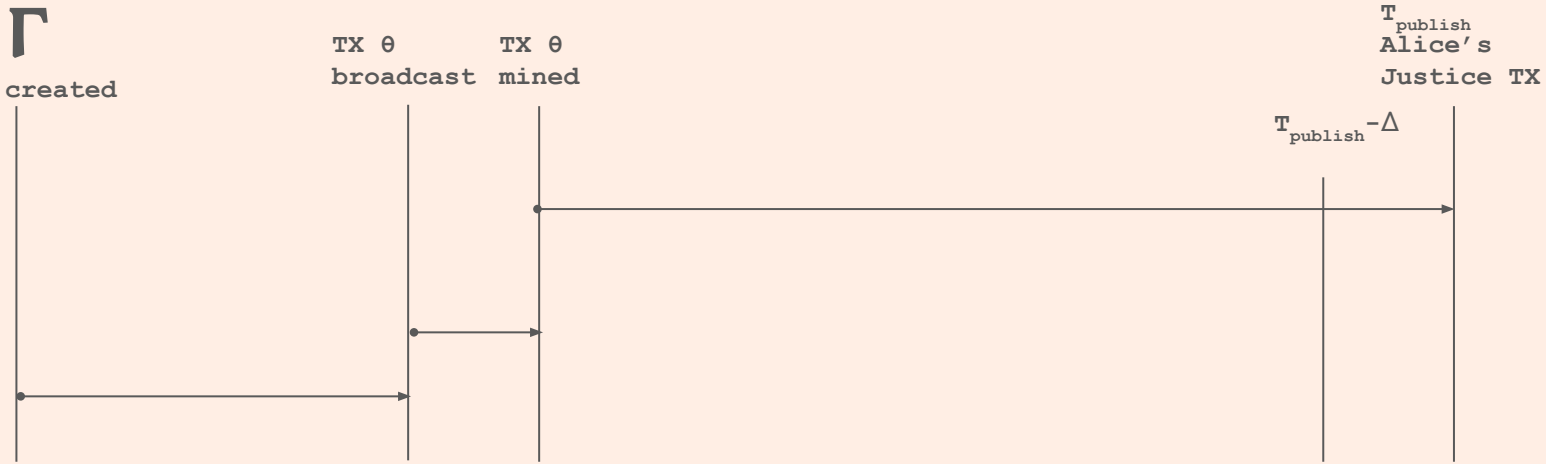
Timing Analysis

Δ = tx inclusion margin



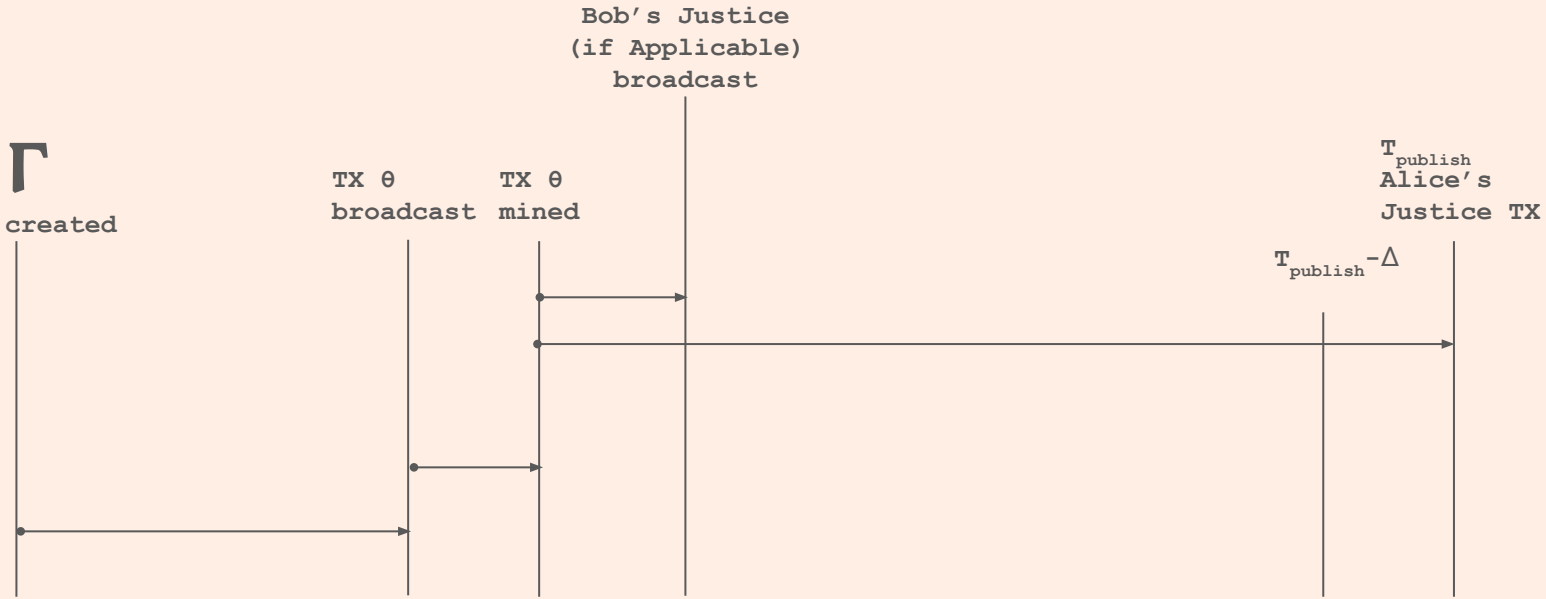
Timing Analysis

Δ = tx inclusion margin



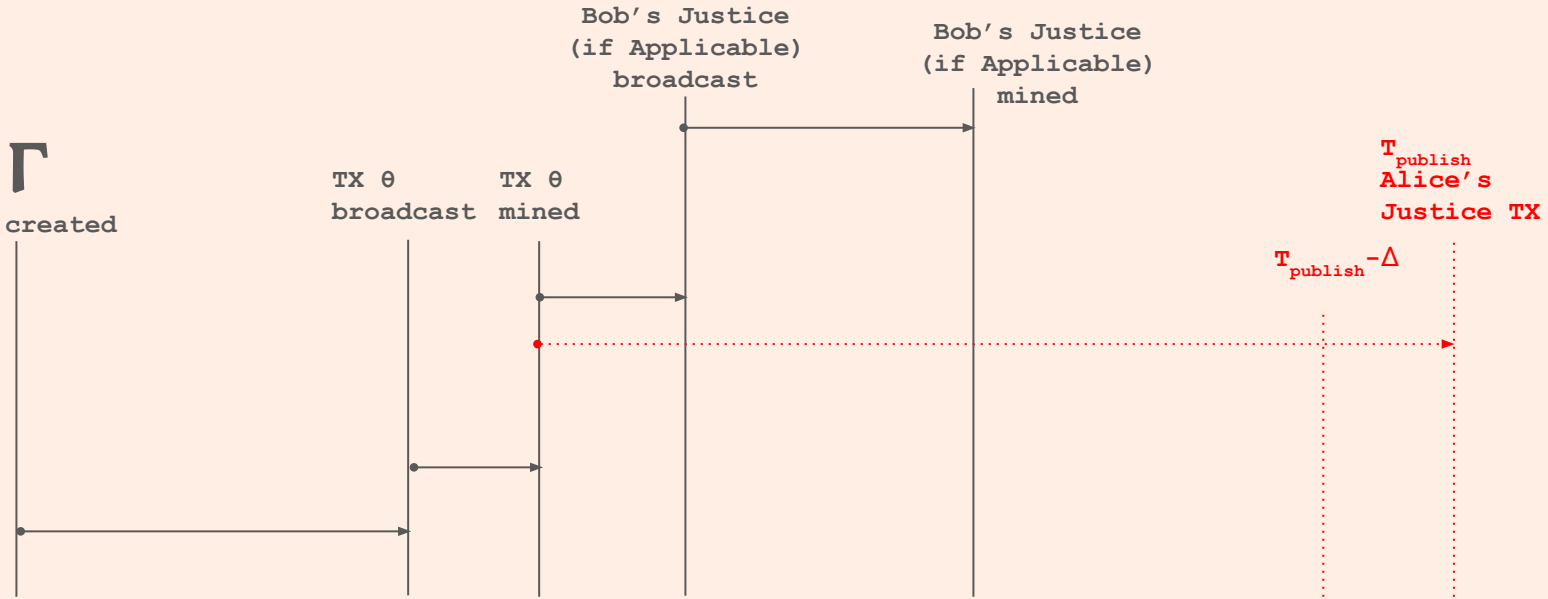
Timing Analysis

Δ = tx inclusion margin



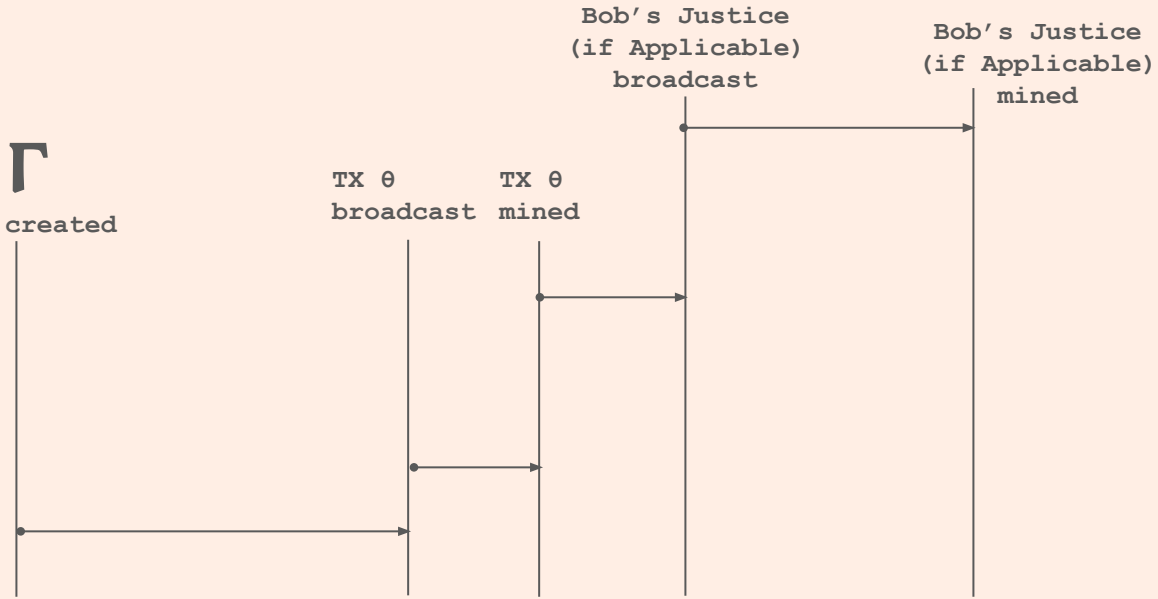
Timing Analysis

Δ = tx inclusion margin



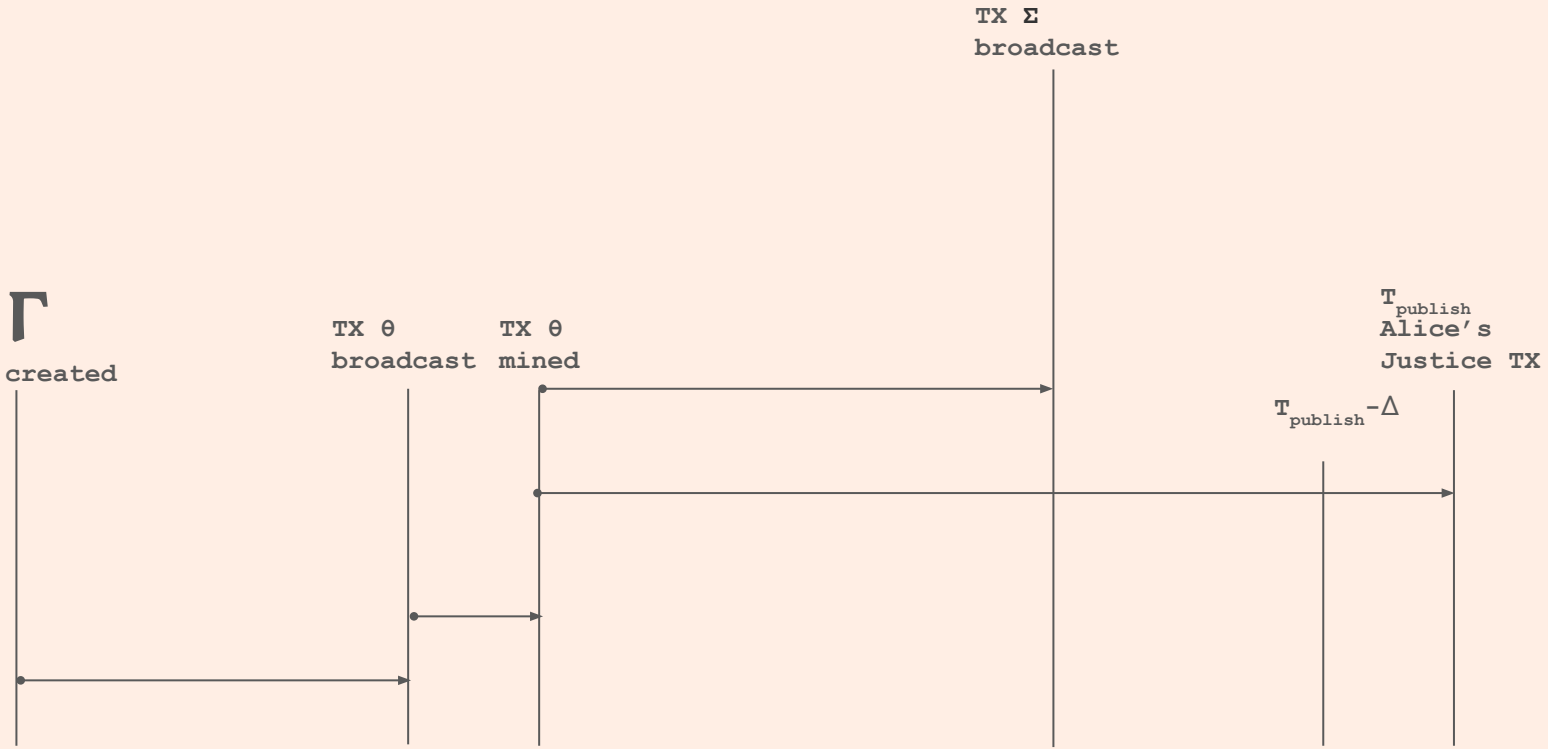
Timing Analysis

Δ = tx inclusion margin



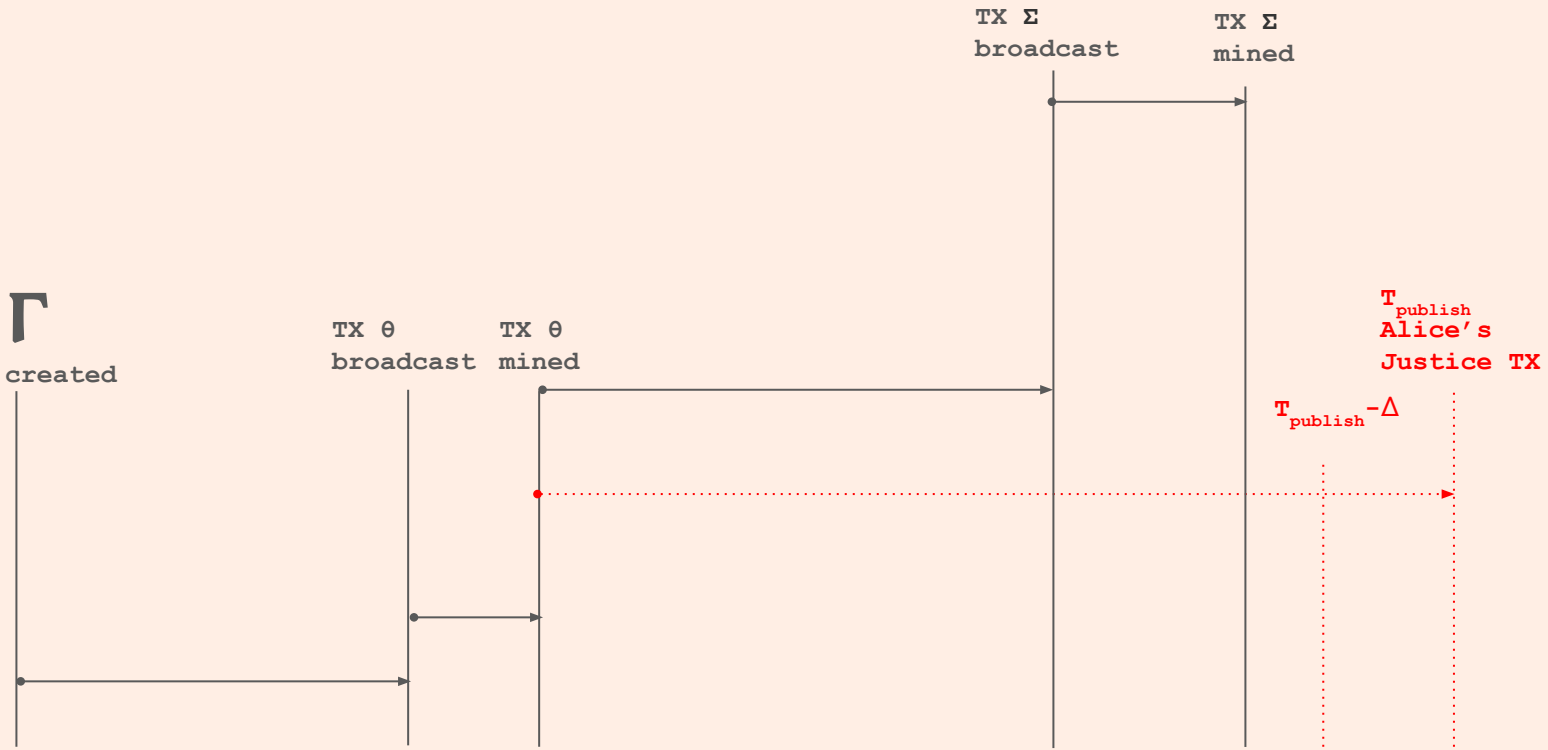
Timing Analysis

Δ = tx inclusion margin



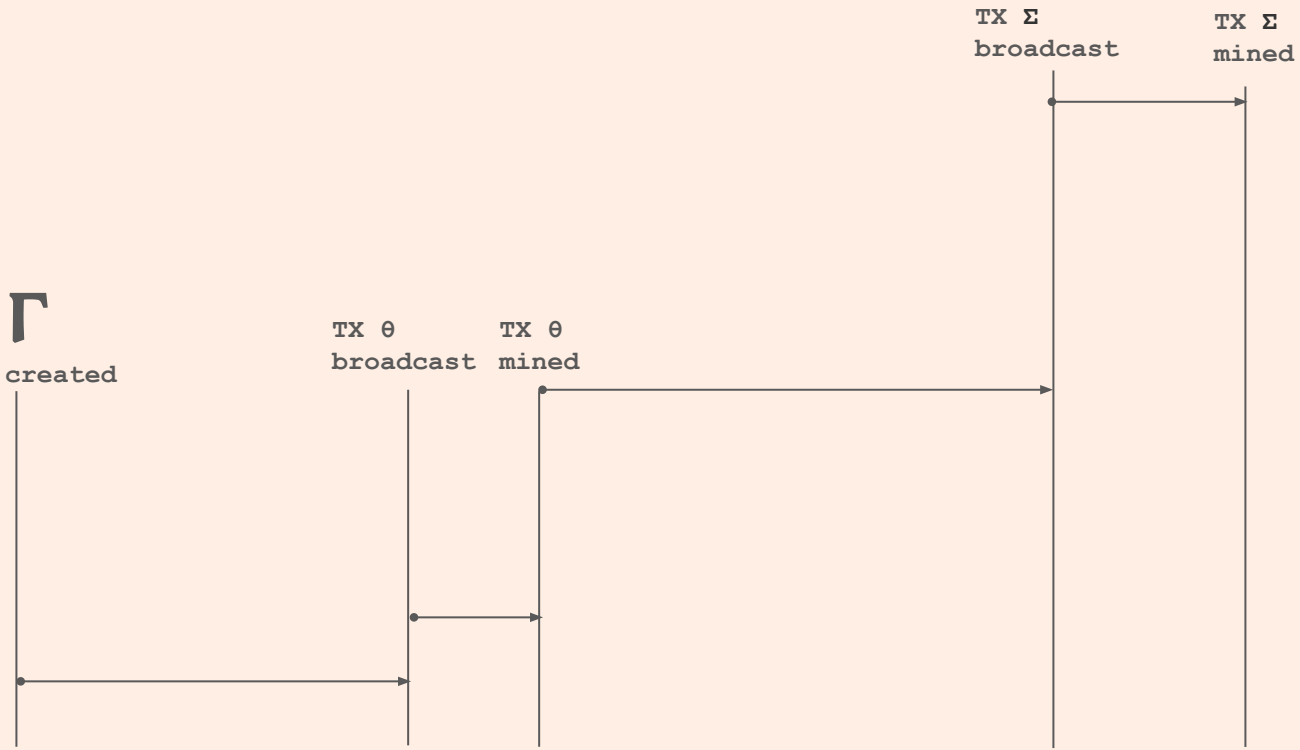
Timing Analysis

Δ = tx inclusion margin



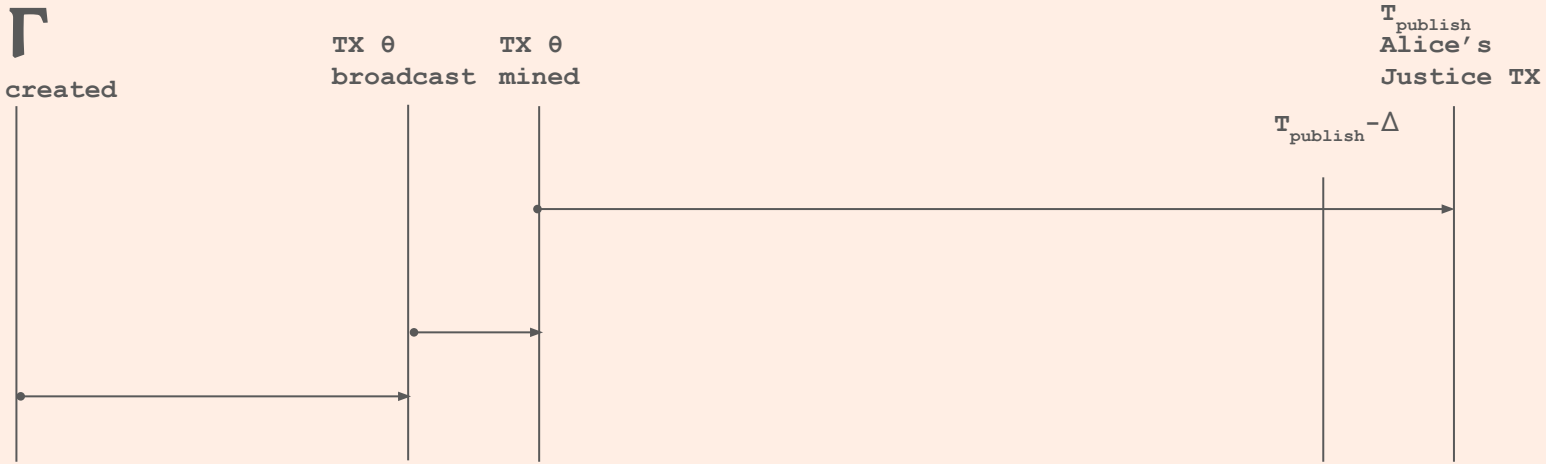
Timing Analysis

Δ = tx inclusion margin



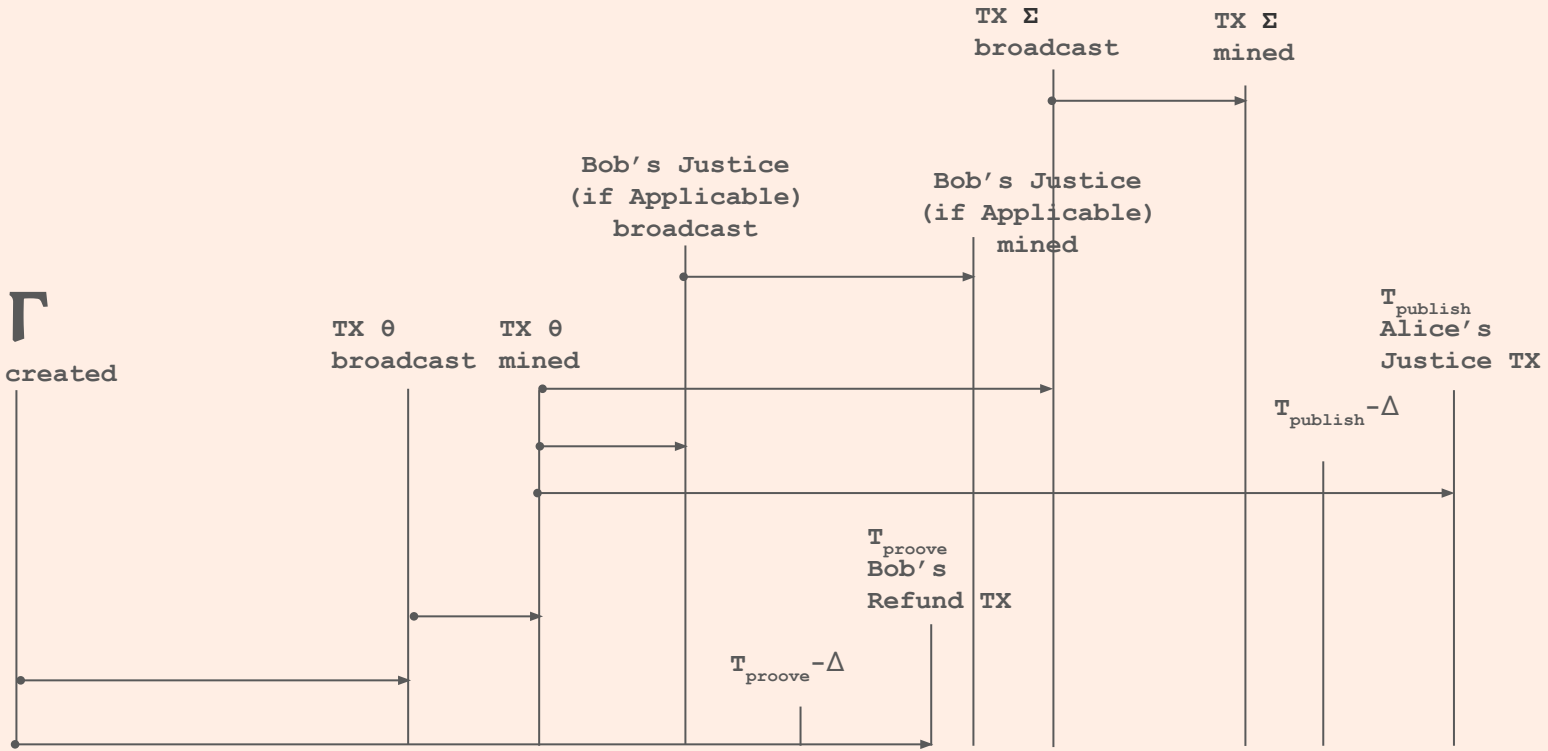
Timing Analysis

Δ = tx inclusion margin



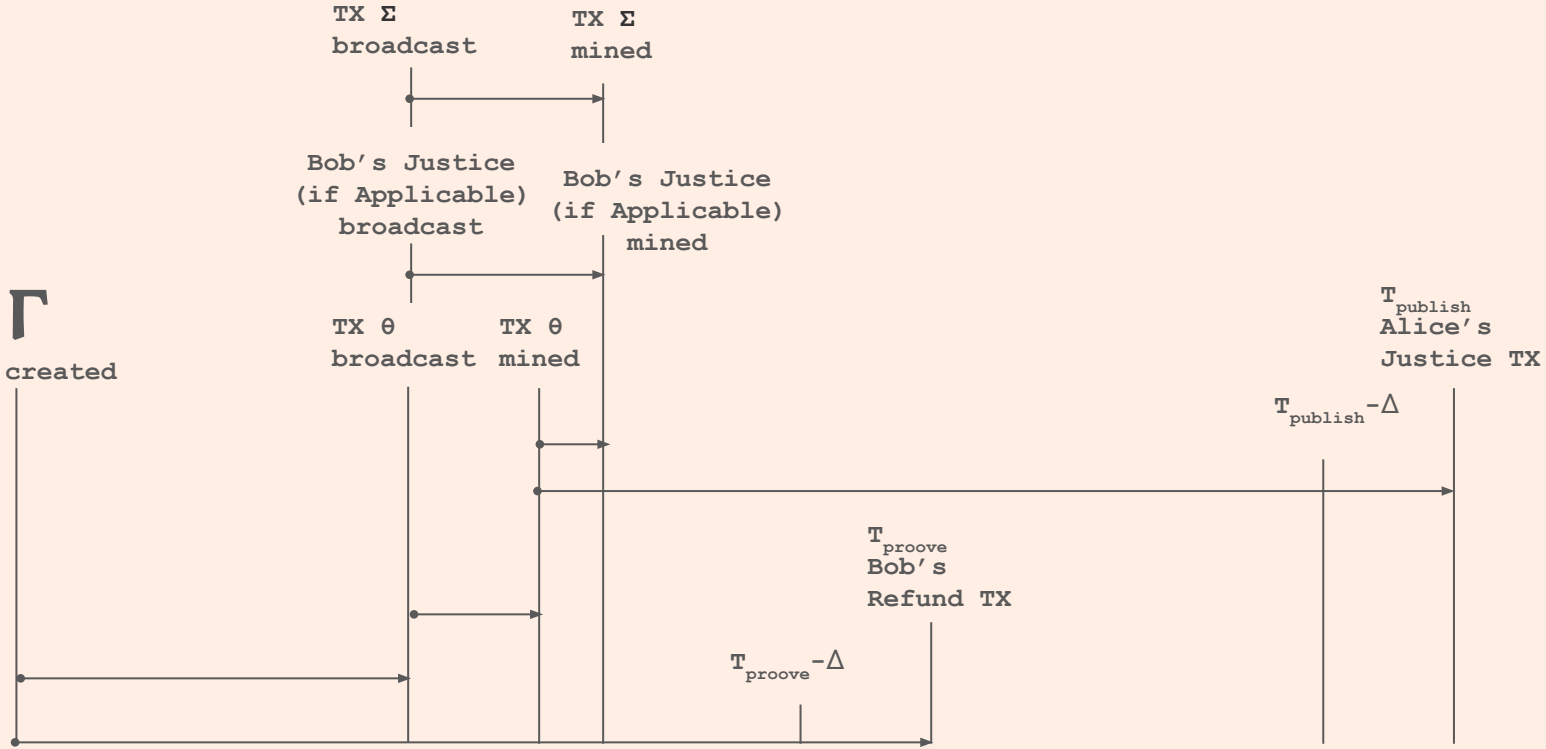
Timing Analysis

Δ = tx inclusion margin



Timing Analysis

Δ = tx inclusion margin



Fund Security Analysis

If Alice “freezes” and never reveals data X:

Bob can claim back the refund TX (all the money or presigned part)

If Bob “freezes” and never finalizes…

Alice can get her refund TX (all the money or presigned default)

Bob punishes Alice if X is invalid (all the money or presigned penalty)

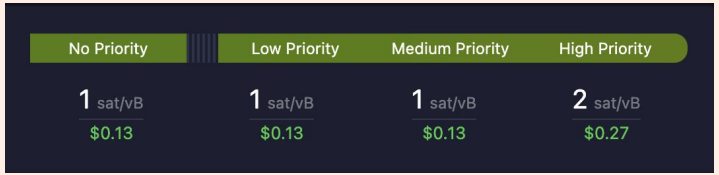
Napkin Math: On-Chain Performance Numbers

Naive Per Gate:

32 + 32 + 32 = 96 bytes per bit of input

200 byte close TX * 8 bits / byte * 96 bytes per bit =

~153,600 bytes witness data



38400 vBytes ⇒ $40 to $80 USD

If willing to lose “1% fee” means min ~$6000 USD / 0.06BTC

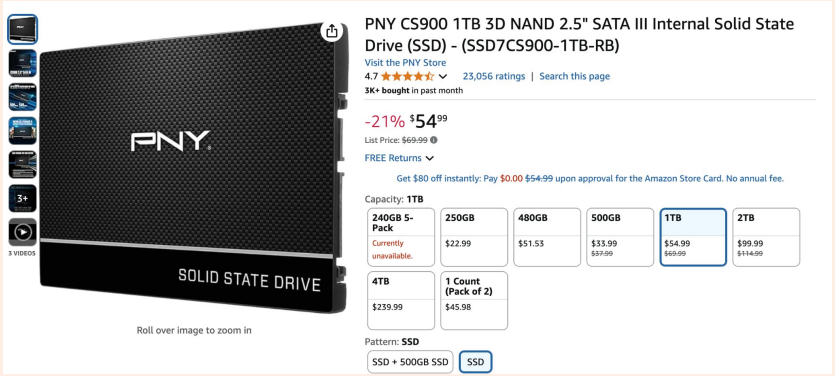
Napkin Math: Off-Chain Numbers

32 x 2 bytes per wire

An average of one ***new*** wire per gate

If a SNARK is 10Bn Gates (seems high)

~ 640 GB of data to exchange off chain ⇒ $50



φ Reexamined

- ZKP Verification, Fixes overall circuit size to constant
- Baking in anti-equivocation

  Rather than an on-chain If X^i_1 and X^i_0 revealed, encrypt

  $$\forall X^i \in X: Y_0 \oplus H(X^i_0 \| X^i_1)$$

- Encrypting **X** so only visible to Alice & Bob

ZK-φ

- Transaction must be plaintext
- Additional ZK proof can show e.g. a valid blockheader with a certain height

Example Application: Bridge

- Bob represents a bridge, and Alice represents a withdrawer. Alice posts collateral for her withdraw request. Alice then requests a withdrawal bound to that txid. Alice then has to provide proof that she started the withdrawal on the sidechain. Bob provides the requested funds. After Alice posts proof, she gets the money.

Example Application: Hashrate Derivatives

Alice and Bob want to make a bet about Bitcoin Hashrate.

Alice bets that if you take run

F(Block(now-100)...Block(now+10,000)), a certain property will or will not hold, and what the distributive tx should be in either case.

Then Alice and Bob fund mutually with C each.

Alice and Bob wait 10k blocks.

Alice then makes a ZK-proof for the dist. tx, and posts-to-close.

The dist said Alice gets 0.9C and Bob gets 1.1 C.

Alice posts proof and waits

- but Bob times out ⇒ Alice claims 2C
- Bob closes ⇒ claims 0.9C, Bob 1.1 C

Example Application: Vault

Bob is a deep cold storage Custodian for Alice.

Alice wants to withdraw money from the cold storage.

Alice requests a withdrawal from the custodial account.

Alice must satisfy the Vaults requirement, which is a proof that no other coins in the vault have been moved in the last month, and that it is authorized by her.

Bob reclaims the funds if Alice can’t prove, and sweeps a penalty.

Example App: Decentralized Hash Buying Pool

Bob is a mining pool operator.

Alice wants to mine to Bob’s pool.

Bob fronts money to a delbrag instance for Alice, along with Alice contributing a “griefing fee”.

Alice mines to an address that Bob made for Alice.

Alice rounds up all the workshares for the period, and proves the total work done and drafts a withdrawal tx.

Alice publishes the withdrawal data.

Bob completes the payment to Alice (or Alice gets all the money).

Example App: Trustless Accelerator

Bob is a Bitcoin Business.

Alice is a miner.

Bob wants to buy a txn SLA from Alice, to guarantee tx inclusion in K blocks, with a term of L>>K blocks.

Bob opens up w/ Alice a Delbrag, he contributes N BTC.

Bob sends Alice a TX F that he wants included, and signs it with the last block hash and hash of history of requests. If it looks good, Alice countersigns.

Alice collects all of his signed receipts and proves if the Txns were mined within the SLA of when Alice signed from the initial block to +L, and proposes Bob’s refund amount.

Example App: Swaps

Alice has some Bitcoin (or a e.g. Token) that Bob wants.

Bob wants to pay for it, but privately.

Bob funds a Delbrag with Alice (and a penalty fee from Alice).

Alice sets up the circuit for a n-deep proof-of-transfer after current height.

Alice pre-proposes a transfer (deviation from normal protocol)

Alice transfers the asset to Bob.

Alice then proves the transfer is mined in a n-deep block.

Alice submits her ZKP on-chain.

Bob either finalizes Alice’s transfer proposal, or shows the proof invalid and reclaims.

Or timeout and Alice claims.

Example App: Garbled Payment Channel

Alice and Bob want to do a payment channel.

They fund a Delbrag mutually.

Alice will commit to, in her data an nLockTime.

Out-of-band, Alice sends Bob new garbled circuits to reveal Y_0 if nLockTime is less than a certain value every time theres an update. (partial re-garbling for efficiency)

Bob keeps just the latest one.

E.g. circuit can only fail if nLockTime < re-garbled(limit)

Basic Extensions

- Cooperative Closing
- Pre-Signed Justice / Refunds

Future Work

- Extending to Multi-Prover Multi-Verifier
- **Γδ**-chains → splitting inputs into **Γ-δ-Γ-δ-Γ-δ-Γ-δ**, in cases where Alice is expected to likely be fraudulent, can reduce data required to be published.
  - These cases occur when an proof-posting party might be "overcomitted" to produce conflicting proofs for different bonds.
  - Useful in Channels where just a bad nLockTime can punish
- Different protocol architectures have different trade-offs (one shot is easy to explain!)
- Hacking Data Availability for smaller proofs (e.g., if input is a ZKP, proving that the redeem tx was inscribed in a block we've seen already)
- **Γ_A-Γ_B-δ** chains, where both parties can publish proof data
- Dynamic "circuit changes" for monotonic properties / keyed data (see channels)

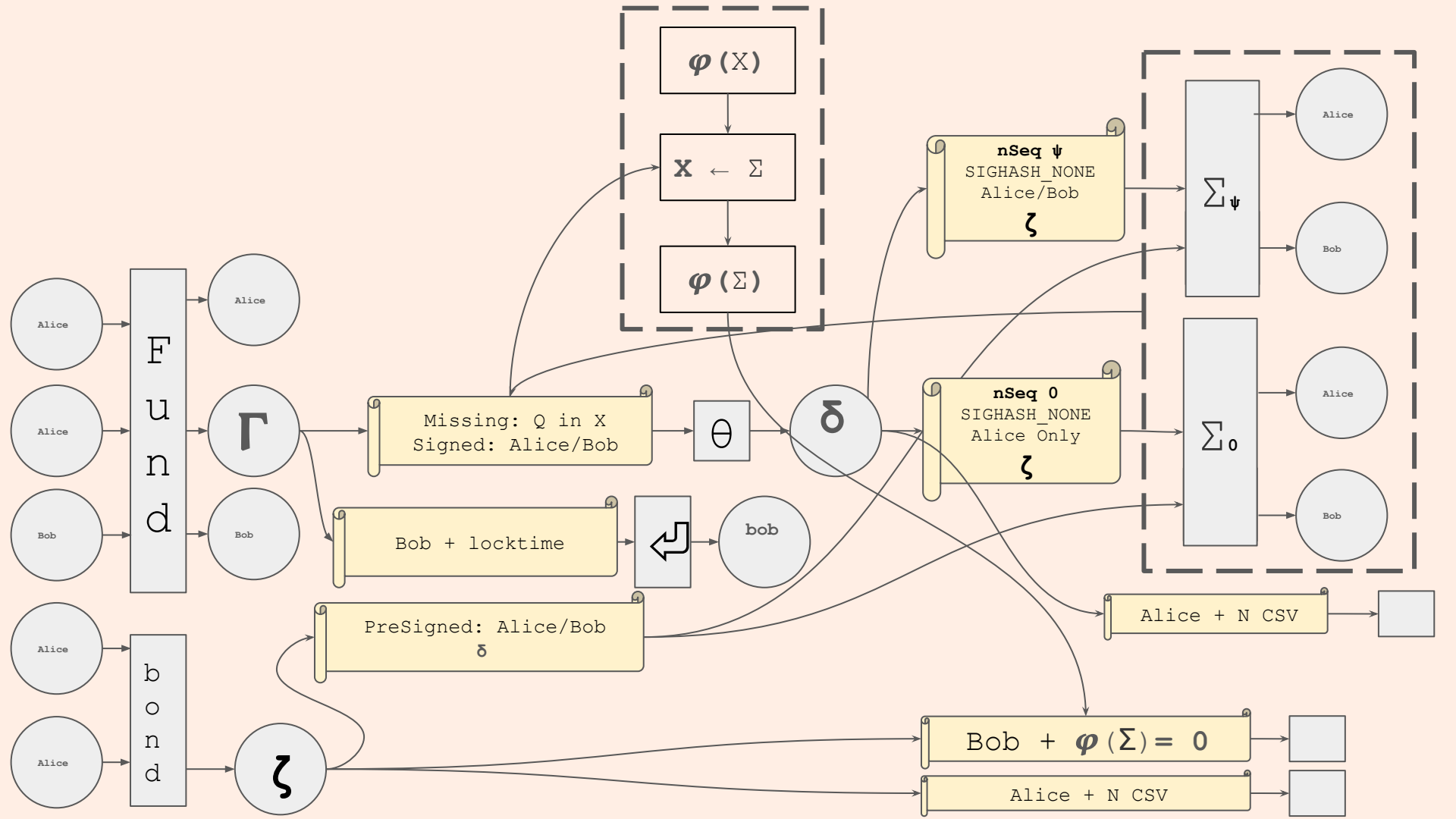
Questions

BACKUP FULL DIAGRAM & Protocol Rev 1

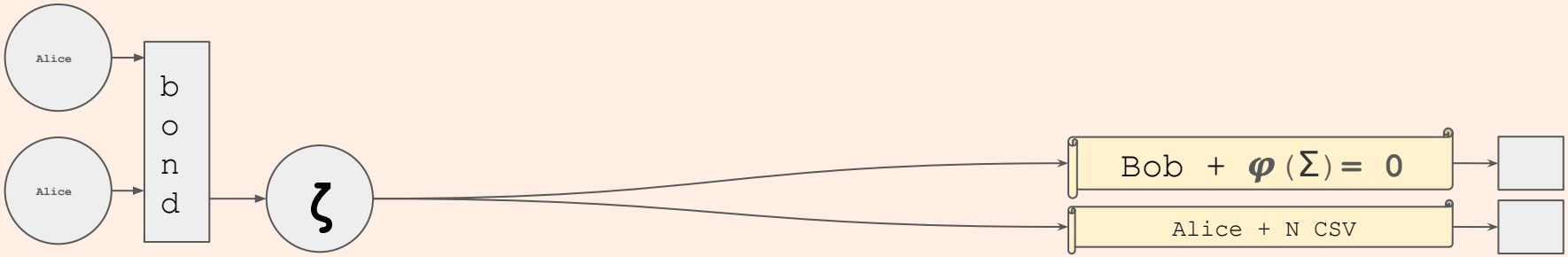
I realized I could make the protocol much simpler for the presentation… But left this for posterity.

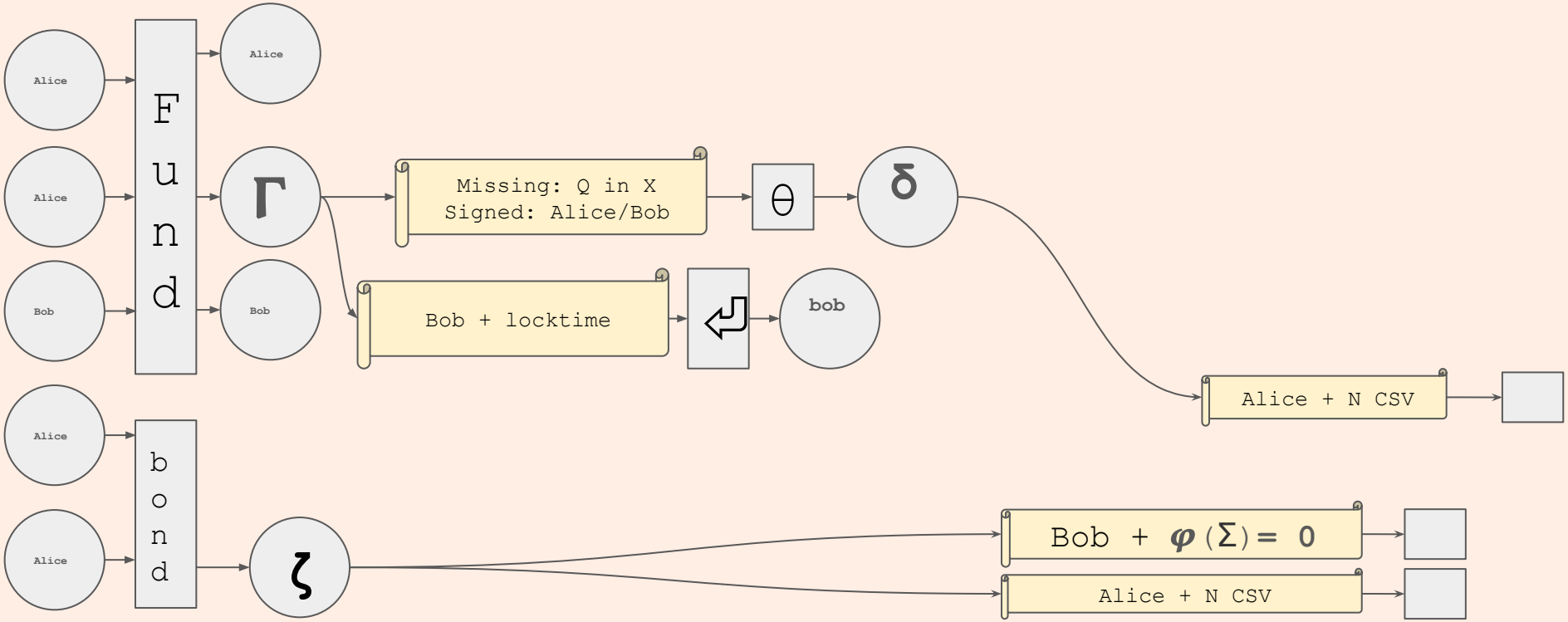
The split input version has uses, too, and some advantages.

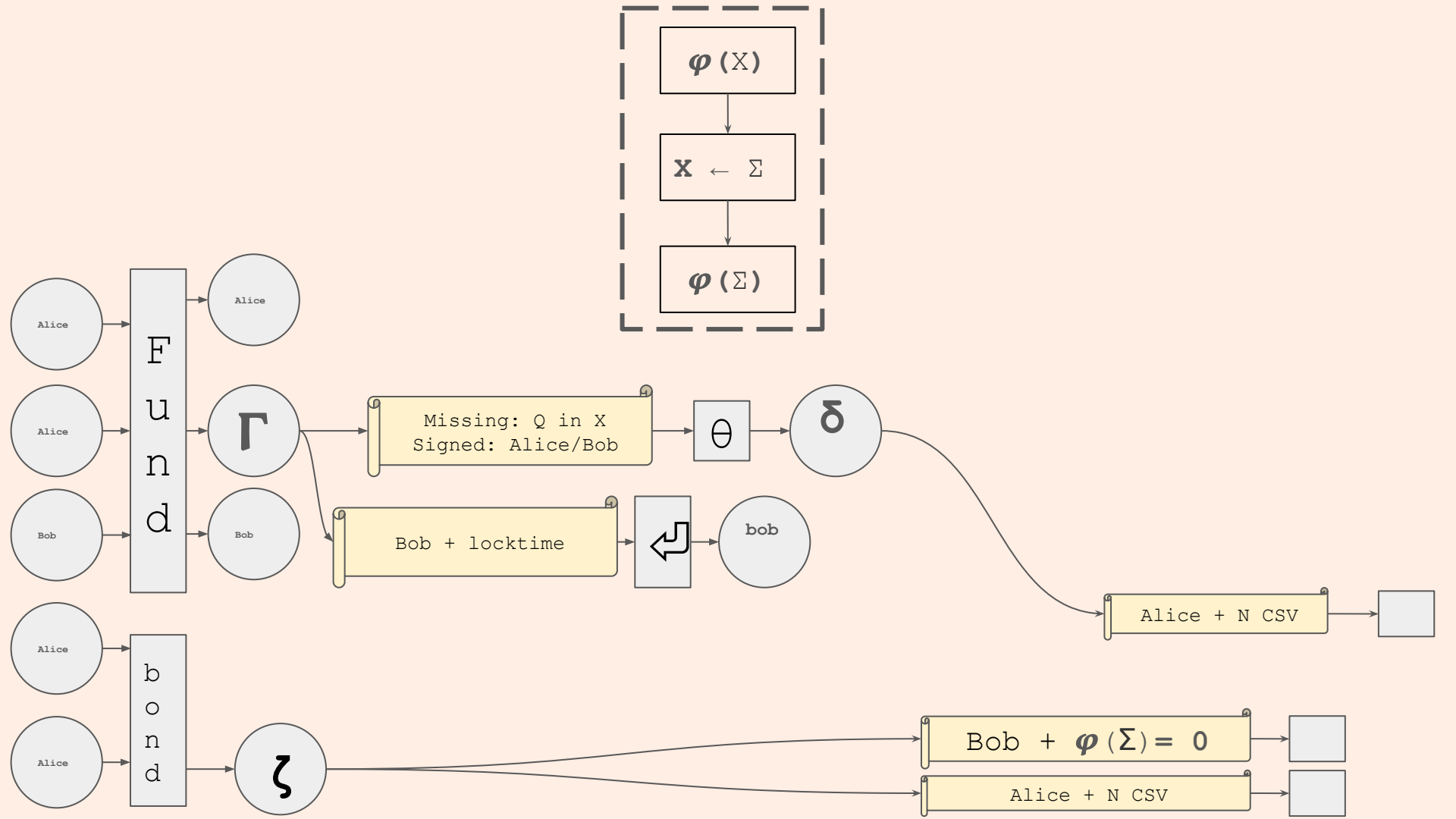
But it’s more complicated.

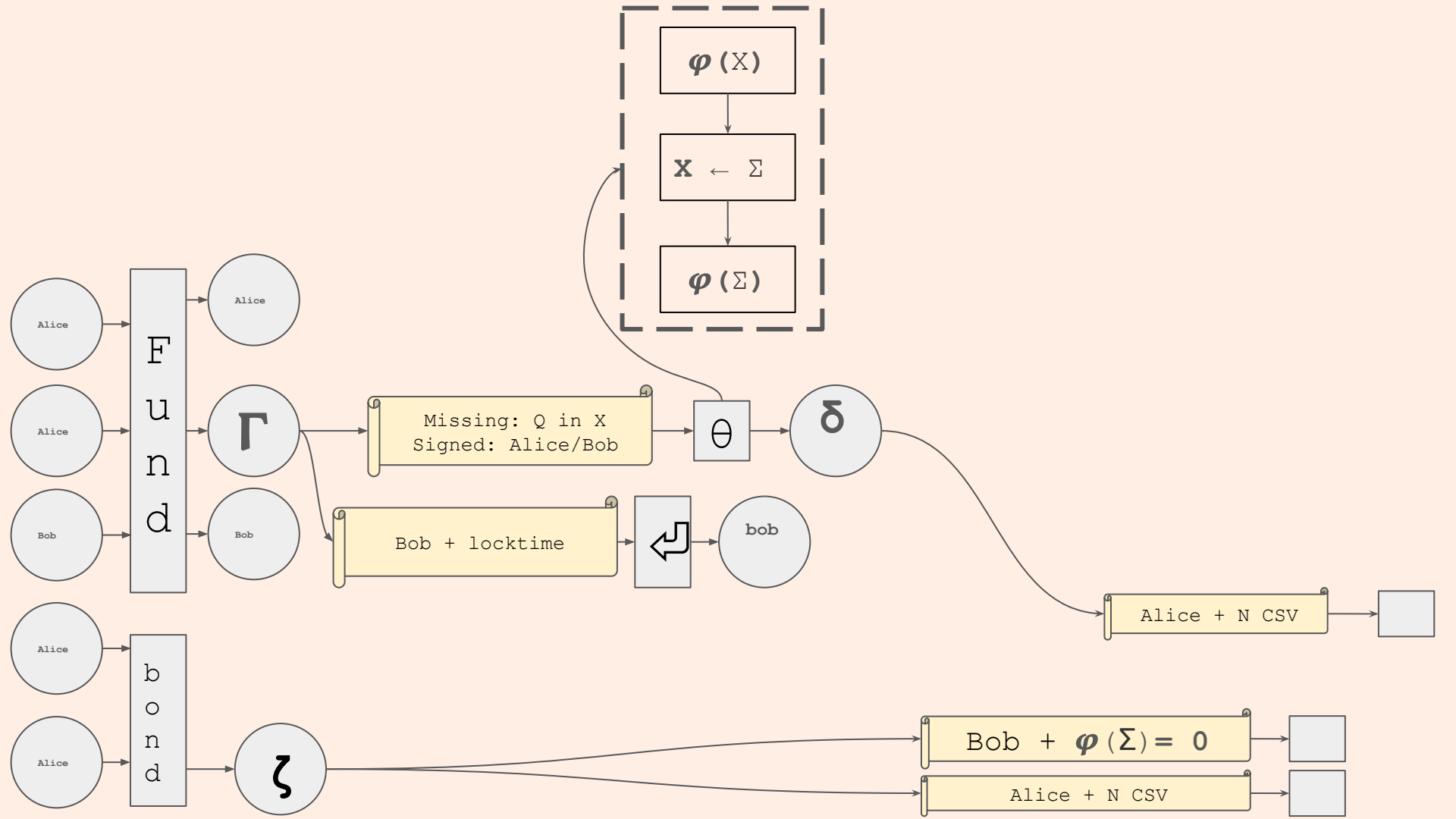

φ(X)
X ← Σ
φ(Σ)
Alice
Alice
Bob
Fund
Alice
Γ
Bob
Missing: Q in X
Signed: Alice/Bob
θ
δ
nSeq ψ
SIGHASH_NONE
Alice/Bob
ζ
nSeq 0
SIGHASH_NONE
Alice Only
ζ
Σψ
Σ0
Alice
Bob
Alice
Bob
Bob + locktime
bob
PreSigned: Alice/Bob
δ
Alice + N CSV
Alice
Alice
bond
ζ
Bob + φ(Σ) = 0
Alice + N CSV

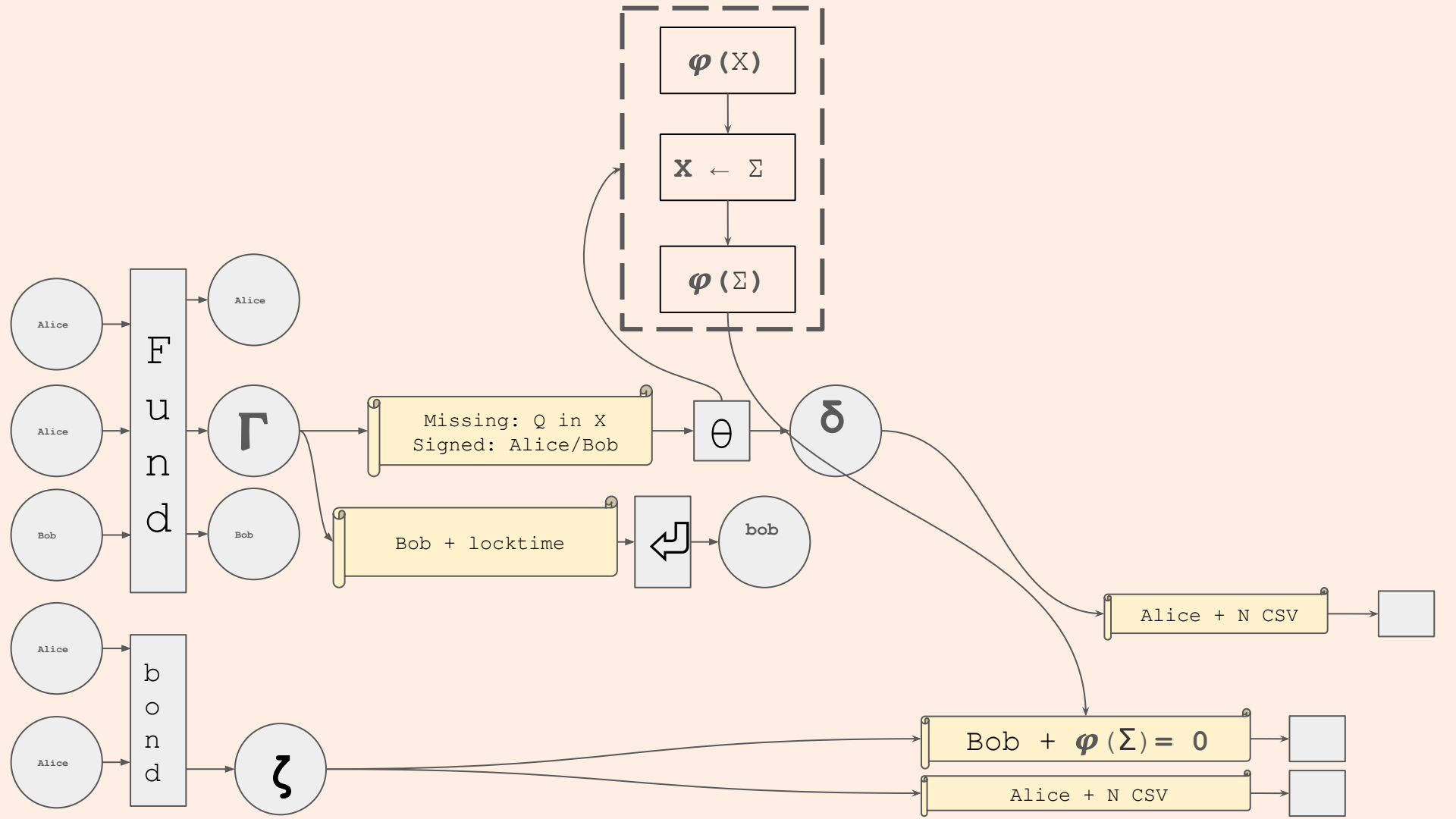

Alice
Alice
bond
ζ
Alice + N CSV

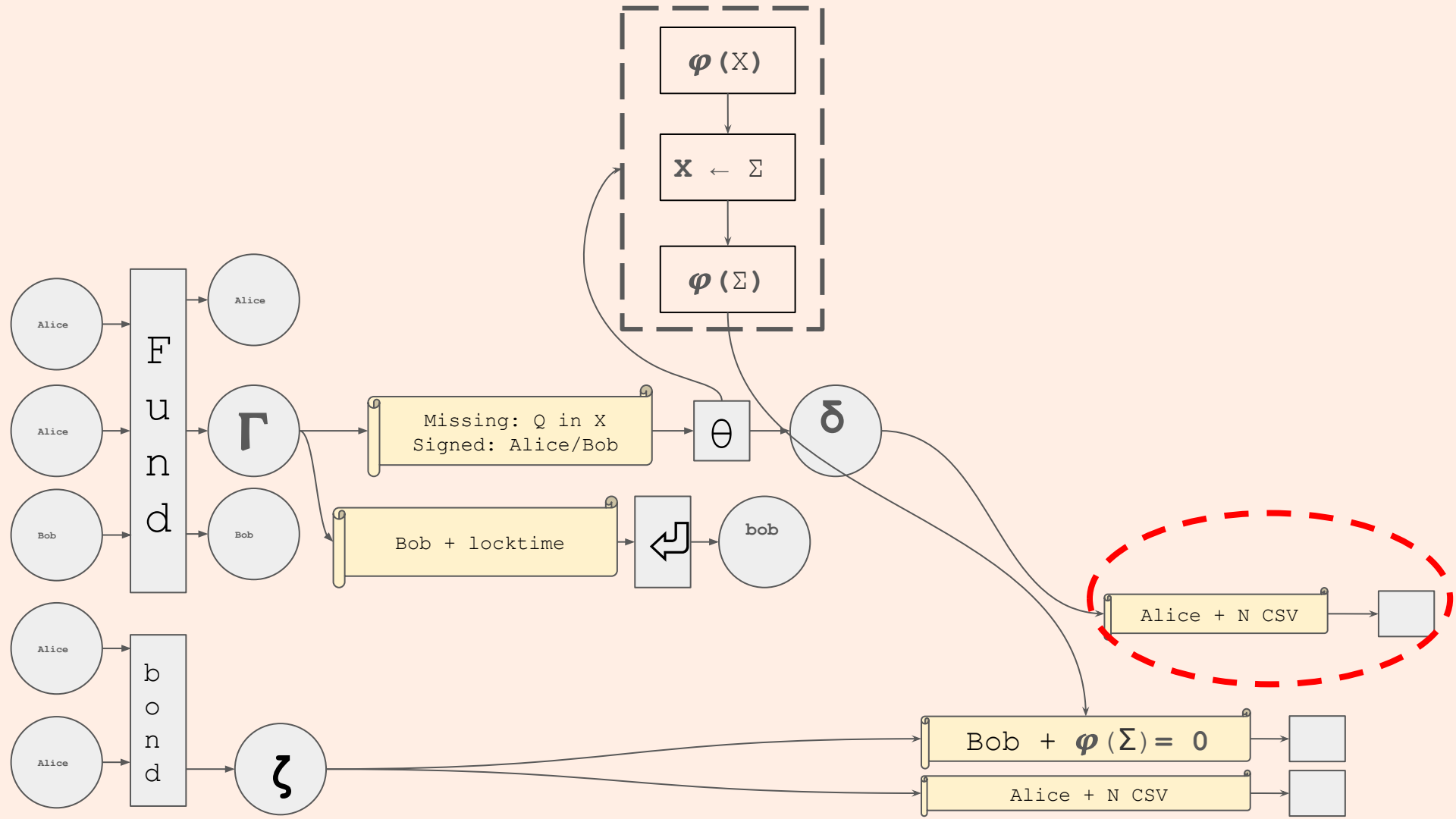

Alice
Alice
bond
ζ
Bob + φ(Σ) = 0
Alice + N CSV


Alice
Alice
Fund
Alice
Alice
Γ
Missing: Q in X
Signed: Alice/Bob
θ
δ
Bob
Bob
Bob + locktime
bob
Alice + N CSV
Alice
Alice
bond
ζ
Bob + φ(Σ) = 0
Alice + N CSV


φ(X)
X ← Σ
φ(Σ)
Alice
Alice
Alice
Fund
Alice
Γ
Bob
Bob
Missing: Q in X
Signed: Alice/Bob
θ
δ
Bob + locktime
bob
Alice + N CSV
Alice
Alice
bond
ζ
Bob + φ(Σ) = 0
Alice + N CSV


φ(X)
x ← Σ
φ(Σ)
Alice
Alice
Alice
Fund
Alice
Γ
Bob
Bob
Missing: Q in X
Signed: Alice/Bob
θ
δ
Bob + locktime
bob
Alice + N CSV
Alice
Alice
bond
ζ
Bob + φ(Σ) = 0
Alice + N CSV

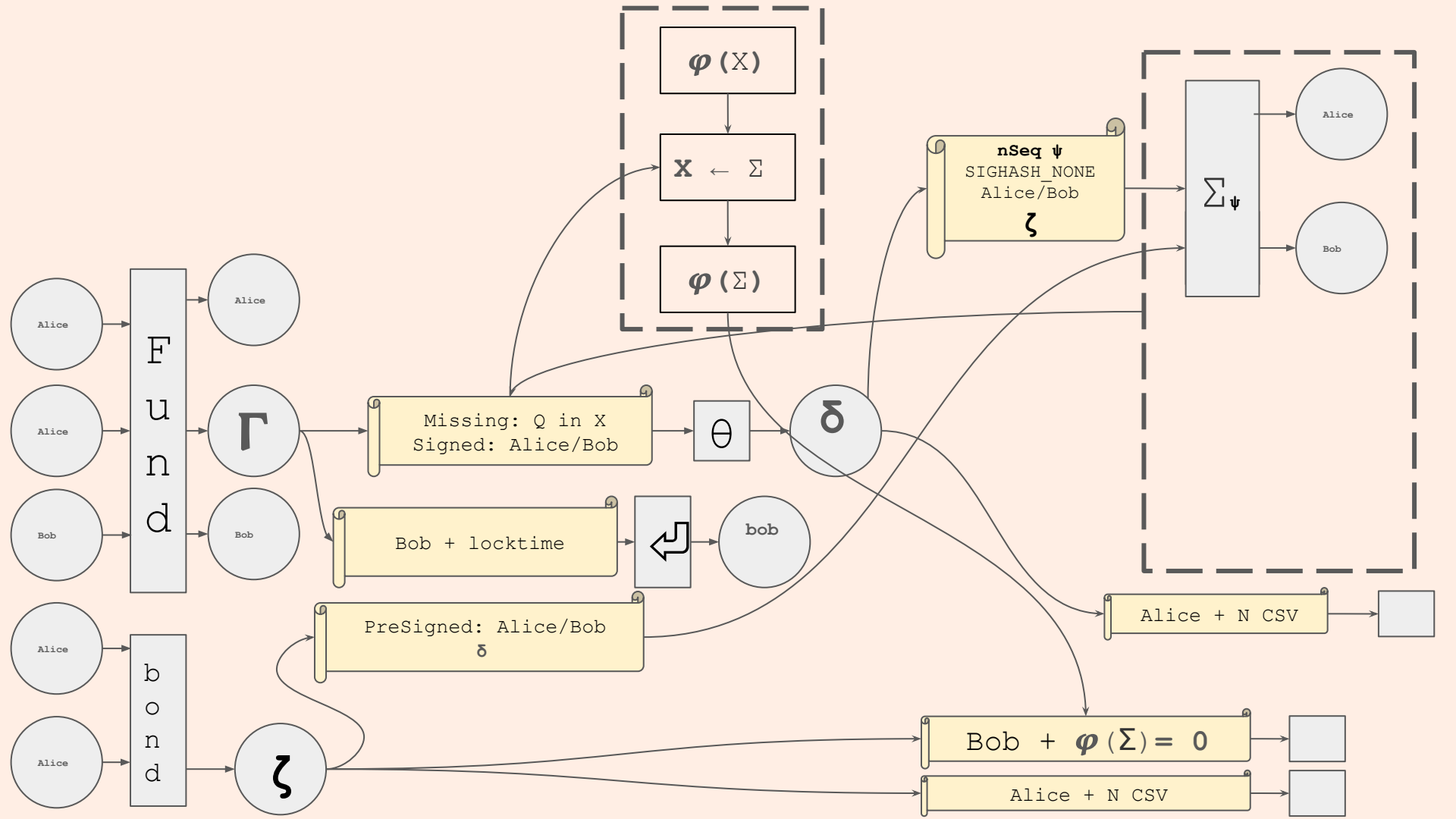

φ(X)
x ← Σ
φ(Σ)
Alice
Alice
Bob
Fund
Alice
Γ
Bob
Missing: Q in X
Signed: Alice/Bob
θ
δ
Bob + locktime
bob
Alice + N CSV
Alice
Alice
bond
ζ
Bob + φ(Σ) = 0
Alice + N CSV

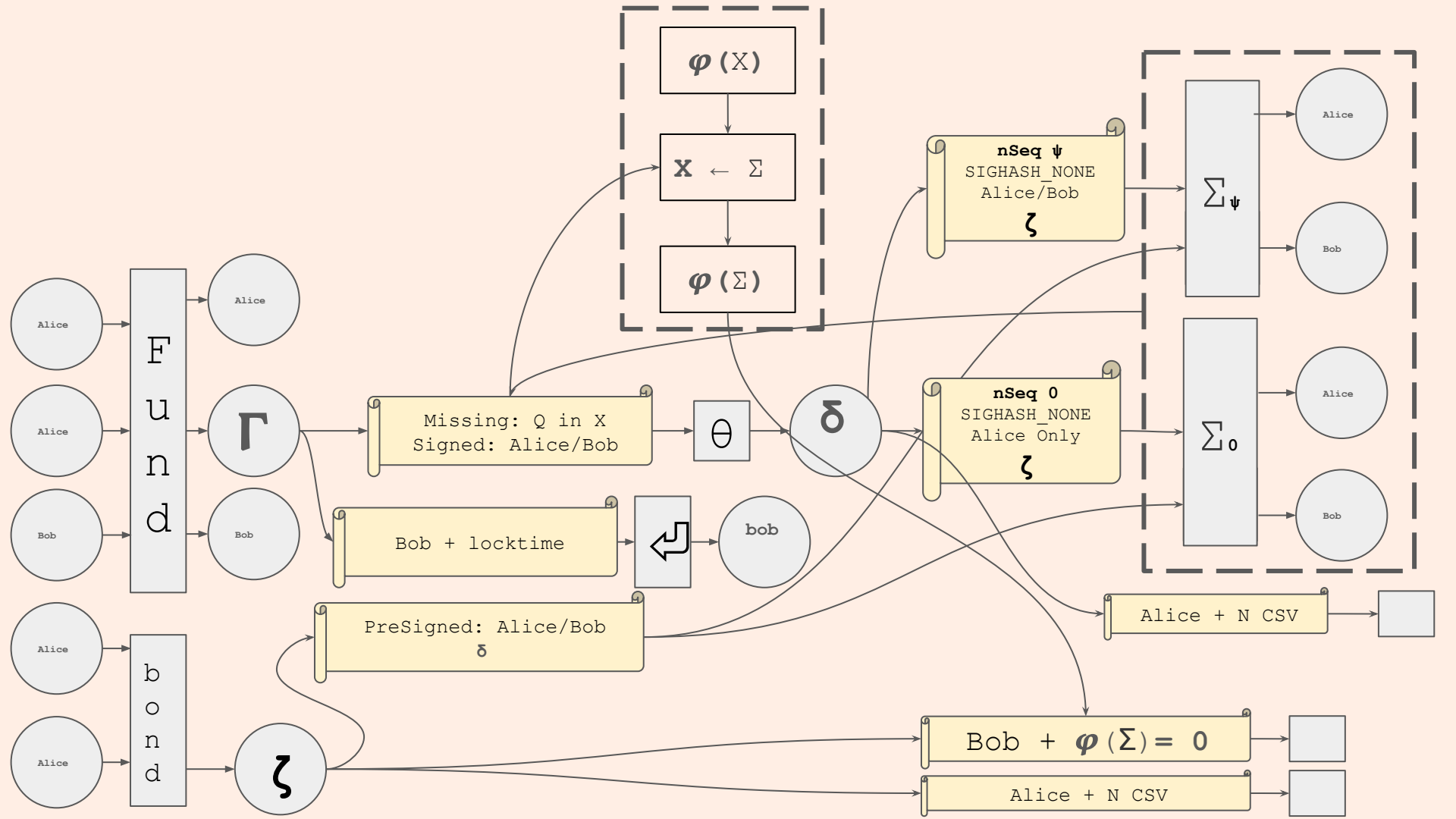

φ(X)
x ← Σ
φ(Σ)
Alice
Alice
Bob
Fund
Alice
Γ
Bob
Missing: Q in X
Signed: Alice/Bob
θ
δ
Bob + locktime
bob
Alice + N CSV
Alice
Alice
bond
ζ
Bob + φ(Σ) = 0
Alice + N CSV

Careful Choice of $\varphi(X)$

X:

Valid transactions redeeming the funds appropriately


φ(X)
X ← Σ
φ(Σ)
nSeq ψ
SIGHASH_NONE
Alice/Bob
ζ
Alice
Bob
Alice
Alice
Bob
Fund
Alice
Γ
Bob
Missing: Q in X
Signed: Alice/Bob
θ
δ
Bob + locktime
bob
PreSigned: Alice/Bob
δ
Alice + N CSV
Alice
Alice
bond
ζ
Bob + φ(Σ) = 0
Alice + N CSV


φ(X)
X ← Σ
φ(Σ)
Alice
Alice
Bob
Fund
Alice
Γ
Bob
Missing: Q in X
Signed: Alice/Bob
θ
δ
nSeq ψ
SIGHASH_NONE
Alice/Bob
ζ
nSeq 0
SIGHASH_NONE
Alice Only
ζ
Σψ
Σ0
Alice
Bob
Alice
Bob
Bob + locktime
bob
PreSigned: Alice/Bob
δ
Alice + N CSV
Alice
Alice
bond
ζ
Bob + φ(Σ) = 0
Alice + N CSV

Timing Analysis

ignoring tx confirm windows