TumbleBit:

An Untrusted Bitcoin-Compatible Anonymous Payment Hub

Ethan Heilman, Leen AlShenibr, Foteini Baldimtsi,
Alessandra Scafuro, Sharon Goldberg

Scaling Bitcoin Milan 2016

BOSTON
UNIVERSITY

Introduction

TumbleBit is:

1. **Private:** Unlinkable Bitcoin payments and k-anonymous mixing,
2. **Untrusted:** No one including Tumbler can steal or link payments.
3. **Scalable (payment hub):** scales transaction velocity and volume.
4. **Compatible:** Works with today's Bitcoin protocol.

Why is compatibility hard?
Our protocol must work with highly constrained Bitcoin scripts which provide two very limited cryptographic operations.

Two ways to use TumbleBit:

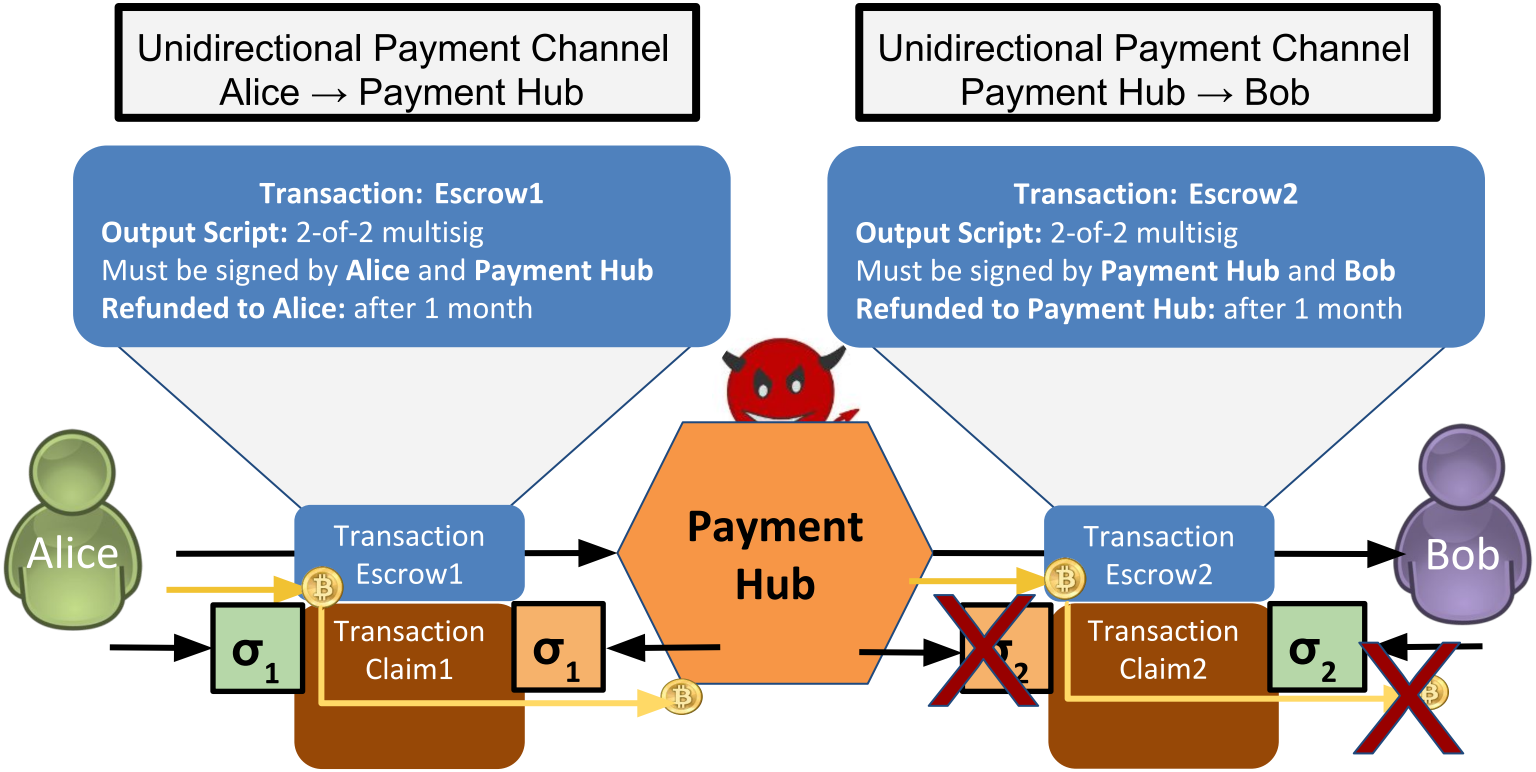
When used as a payment hub, TumbleBit helps scale Bitcoin's transaction velocity (faster payments), and transaction volume (higher maximum payments).

When TumbleBit is used as a payment hub:

- Unlinkability within the payment phase,
- Payments confirmed in seconds,
- Payments are off-blockchain,
  ... don't take up space on the blockchain.

Background: Payment Hub

A payment hub: routes payment channels.



Unidirectional Payment Channel
Alice → Payment Hub

Transaction: Escrow1
Output Script: 2-of-2 multisig
Must be signed by **Alice** and **Payment Hub**
Refunded to Alice: after 1 month

Unidirectional Payment Channel
Payment Hub → Bob

Transaction: Escrow2
Output Script: 2-of-2 multisig
Must be signed by **Payment Hub** and **Bob**
Refunded to Payment Hub: after 1 month

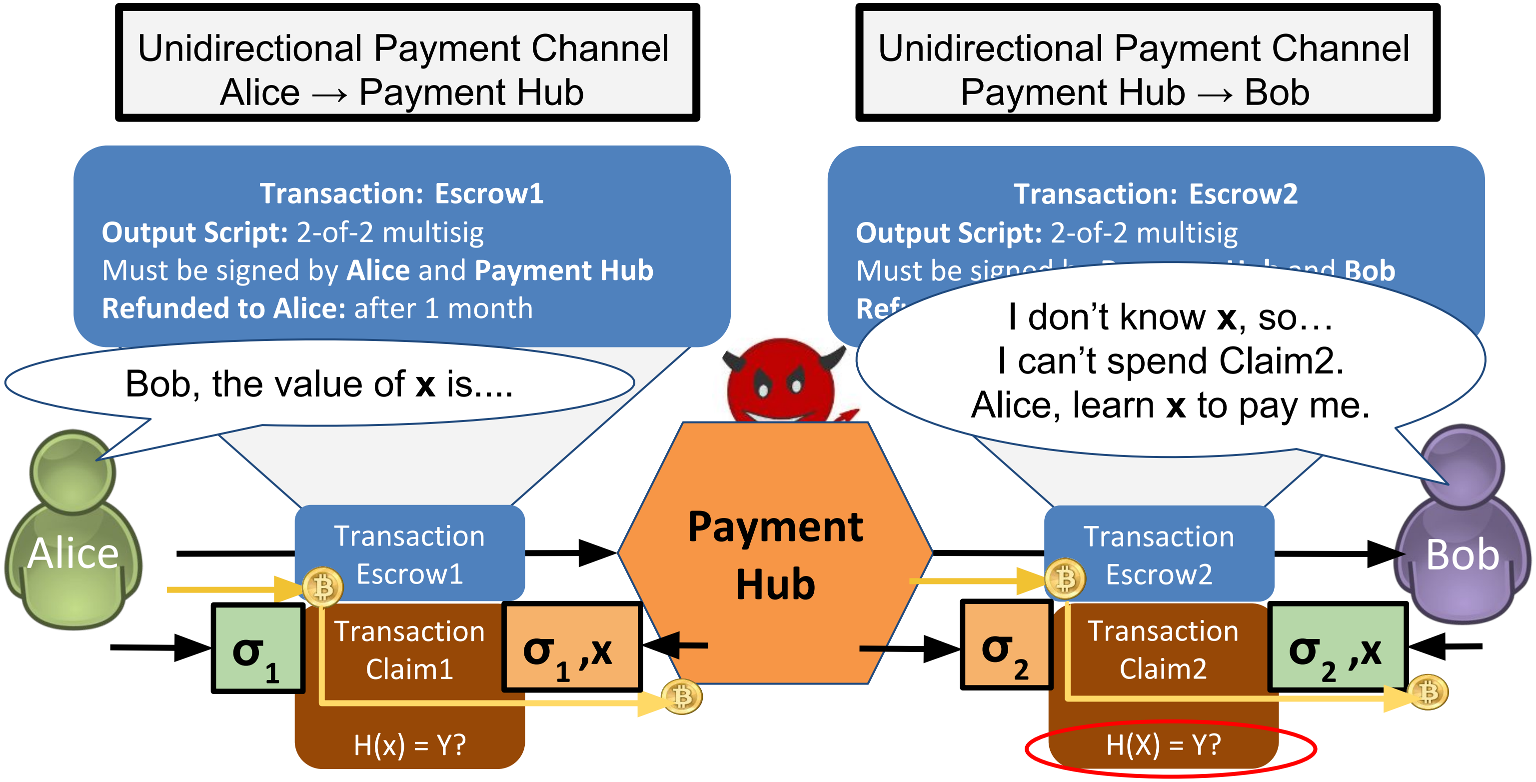
...But what if the hub is malicious,

Atomicity: If Claim1 and Claim2 happen atomically then theft is prevented.

Hash locks provide this property.

Background: Payment Hub

A payment hub: routes payment channels.



Unidirectional Payment Channel
Alice → Payment Hub

Unidirectional Payment Channel
Payment Hub → Bob

Transaction: Escrow1
Output Script: 2-of-2 multisig
Must be signed by **Alice** and **Payment Hub**
Refunded to Alice: after 1 month

Transaction: Escrow2
Output Script: 2-of-2 multisig
Must be signed and **Bob**
Ref

Bob, the value of **x** is....

I don't know **x**, so…
I can't spend Claim2.
Alice, learn **x** to pay me.

Alice

Payment Hub

Bob

Transaction Escrow1

Transaction Claim1

σ_1

σ_1,x

H(x) = Y?

Transaction Escrow2

Transaction Claim2

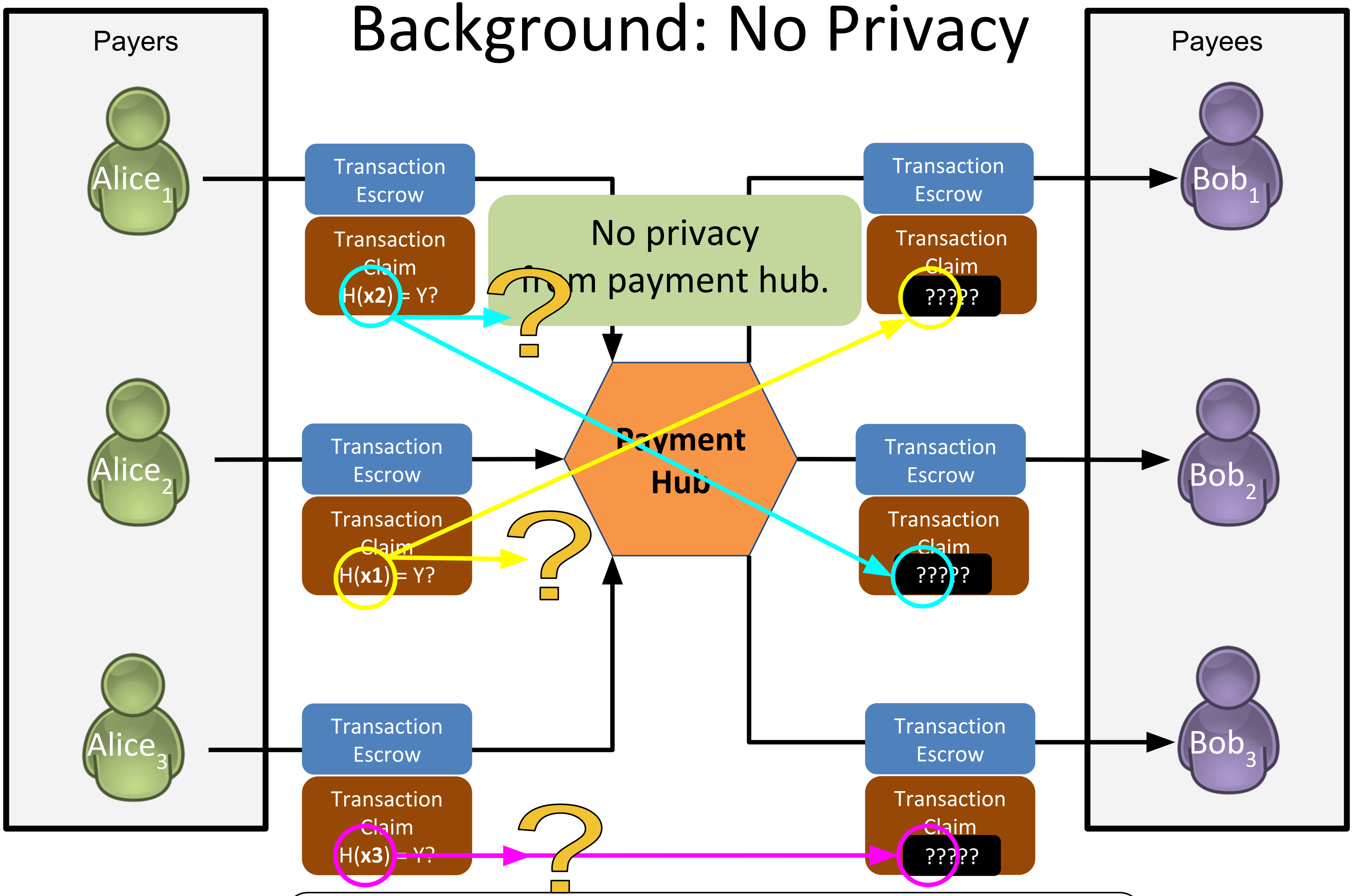
σ_2

σ_2,x

H(X) = Y?

Thus, using hash locked transactions or HTLCs a payment hub can prevent theft, however this is provides no privacy against the payment hub.

Background: No Privacy



Payers

Alice$_1$

Alice$_2$

Alice$_3$

Transaction Escrow

Transaction Claim

H(**x2**) = Y?

Transaction Escrow

Transaction Claim

H(**x1**) = Y?

Transaction Escrow

Transaction Claim

H(**x3**) = Y?

No privacy from payment hub.

Payment Hub

Payees

Bob$_1$

Bob$_2$

Bob$_3$

Transaction Escrow

Transaction Claim

?????

Transaction Escrow

Transaction Claim

?????

Transaction Escrow

Transaction Claim

?????

The main idea behind TumbleBit is a protocol which provides **atomicity** but is also **unlinkable** (i.e. private). Think of it like Unlinkable or Private HTLCs.

RSA Puzzles

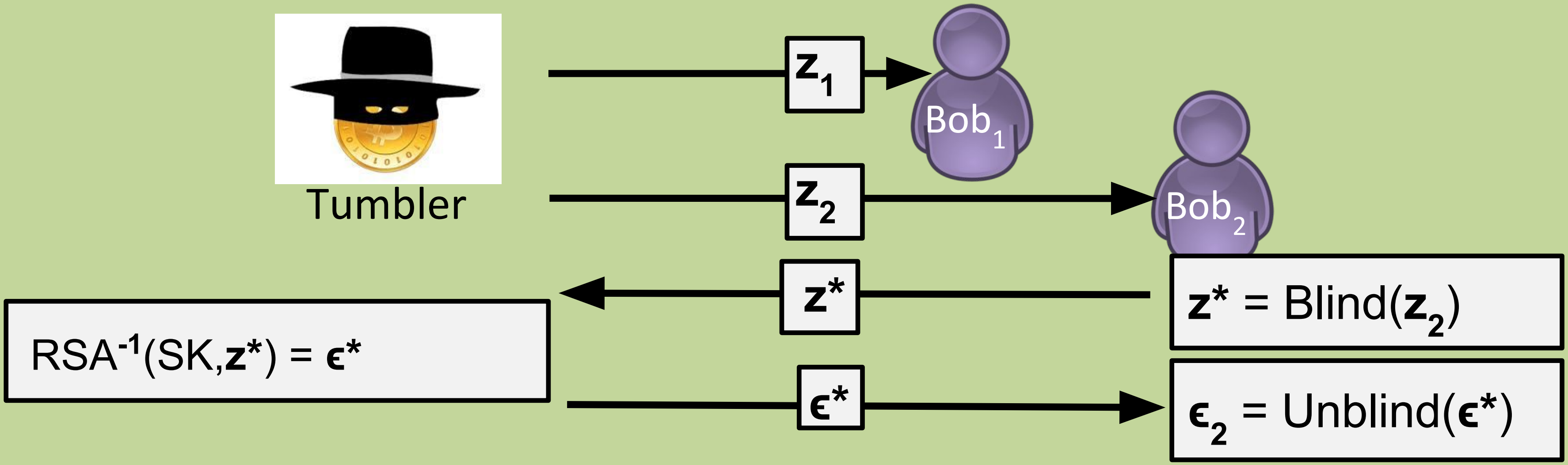
- An RSA Puzzle is just a “textbook RSA encryption” of some value **ϵ**:

$$\mathrm{RSA}(\mathrm{PK}, \boldsymbol{\epsilon}) = \mathbf{z}$$

- Only the party that knows SK can solve RSA puzzles**:**

$$\mathrm{RSA}^{-1}(\mathrm{SK}, \mathbf{z}) = \mathrm{RSA}^{-1}(\mathrm{SK}, \mathrm{RSA}(\mathrm{PK}, \boldsymbol{\epsilon})) = \boldsymbol{\epsilon}$$

RSA blinding can be used to blind RSA puzzles



$\mathbf{z}^* = \mathrm{Blind}(\mathbf{z}_2)$

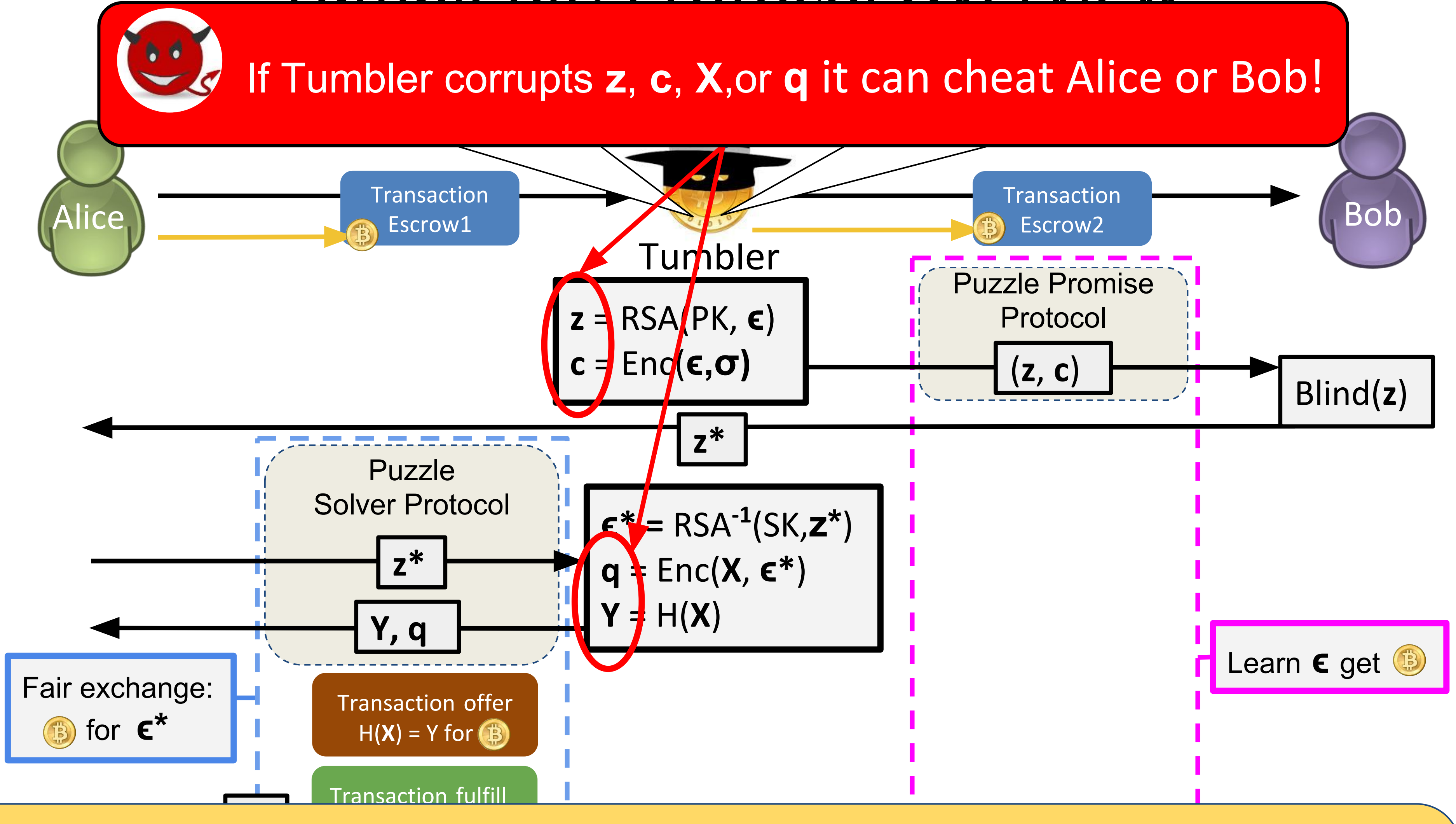
$\mathrm{RSA}^{-1}(\mathrm{SK},\mathbf{z}^*) = \boldsymbol{\epsilon}^*$

$\boldsymbol{\epsilon}_2 = \mathrm{Unblind}(\boldsymbol{\epsilon}^*)$

Bob$_2$ learns the solution $\boldsymbol{\epsilon}_2$ to the puzzle $\mathbf{z}_2$

Tumbler can not link the blinded RSA puzzle it solves $\mathbf{z}^*$
to any of the RSA puzzles it issued ($\mathbf{z}_1$, $\mathbf{z}_2$).

TumbleBit: Protocol Overview



If Tumbler corrupts **z**, **c**, **X**,or **q** it can cheat Alice or Bob!

TumbleBit prevents this via two protocols:

Puzzle-Solver-Protocol:
Tumbler convinces Alice the preimage **X** where Hash(**X**) = **Y** will allow her to learn **ϵ***.

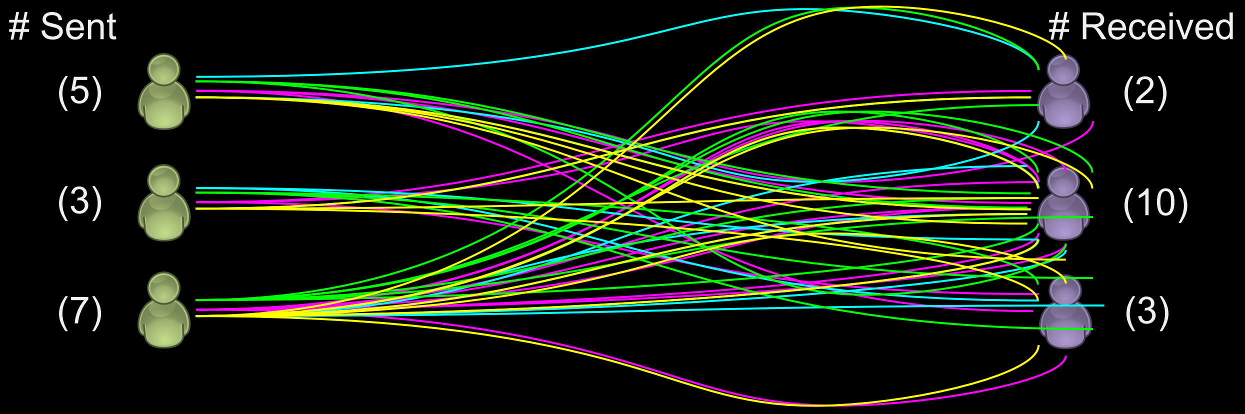
Puzzle-Promise-Protocol:
Tumbler convinces Bob that the solution to RSA puzzle **z** is a value **ϵ** which allows him learn **σ**.

TumbleBit: Phases

Privacy offered the TumbleBit Payment Hub

Tumbler’s view:
(1) payer of each payment, (2) # of payments each payee received.

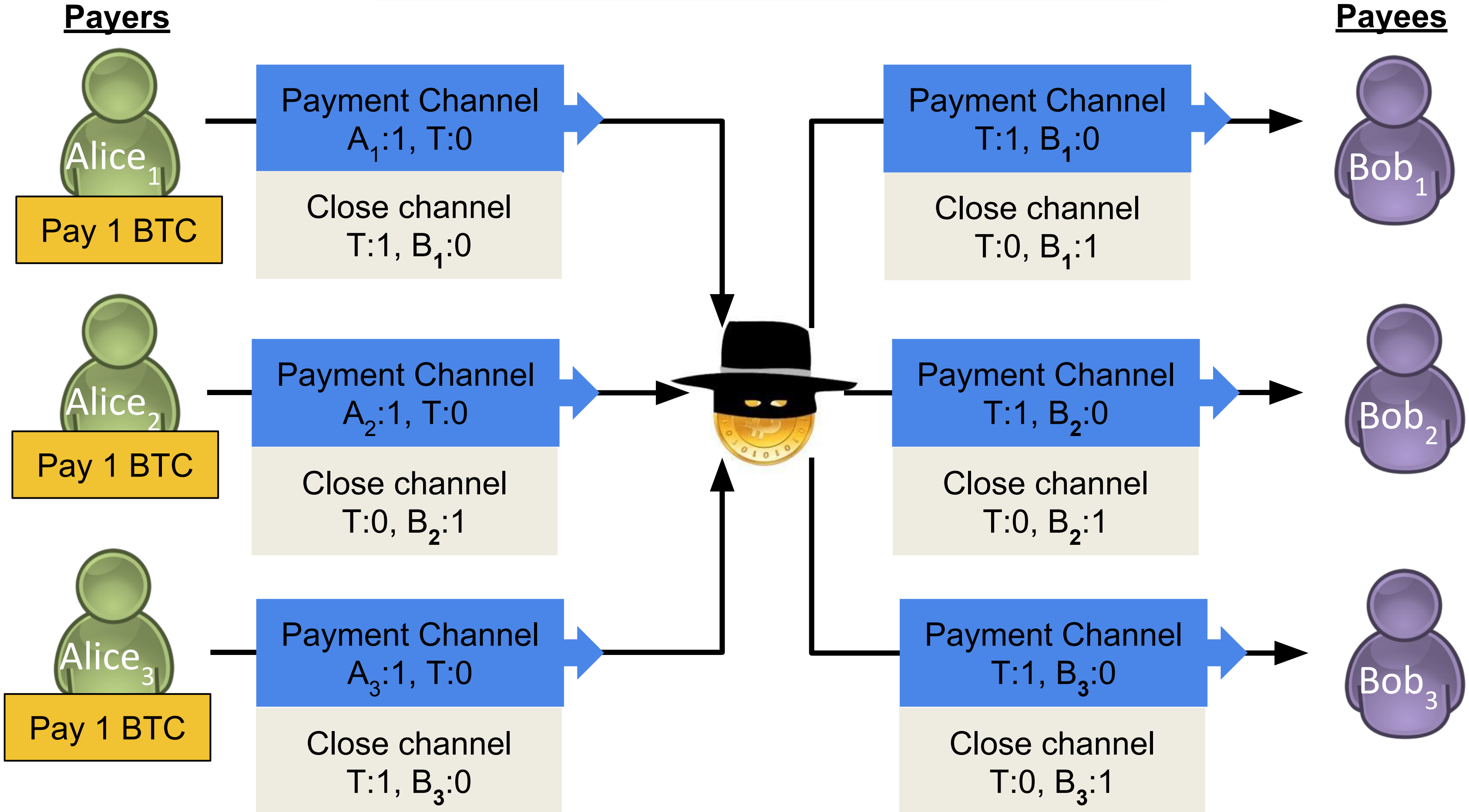
Unlinkability def:
All interaction graphs compatible with the tumblers view are equally likely.



TumbleBit: Classic Tumbler

To run TumbleBit as a Classic Bitcoin Tumbler:

- Each payer just makes one payment.
- Each payee accepts only one payment.
- # of payers = # of payees.



Payers

Alice$_1$

Pay 1 BTC

Payment Channel
A_1:1, T:0

Close channel
T:1, B_1:0

Alice$_2$

Pay 1 BTC

Payment Channel
A_2:1, T:0

Close channel
T:0, B_2:1

Alice$_3$

Pay 1 BTC

Payment Channel
A_3:1, T:0

Close channel
T:1, B_3:0

Payees

Payment Channel
T:1, B_1:0

Close channel
T:0, B_1:1

Bob$_1$

Payment Channel
T:1, B_2:0

Close channel
T:0, B_2:1

Bob$_2$

Payment Channel
T:1, B_3:0

Close channel
T:0, B_3:1

Bob$_3$

Provides k-anonymity:

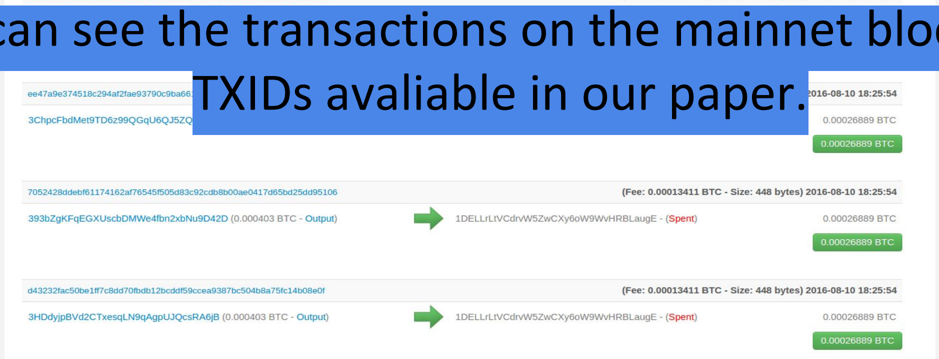
Where k = # of payers = # of payee.

TumbleBit: Implementation

We wrote a proof-of-concept implementation of the Tumbler mode:

- We are working on improving it and making it user friendly.
- Sourcecode and a development roadmap are available on Github.

We “tumbled” 800 payments:



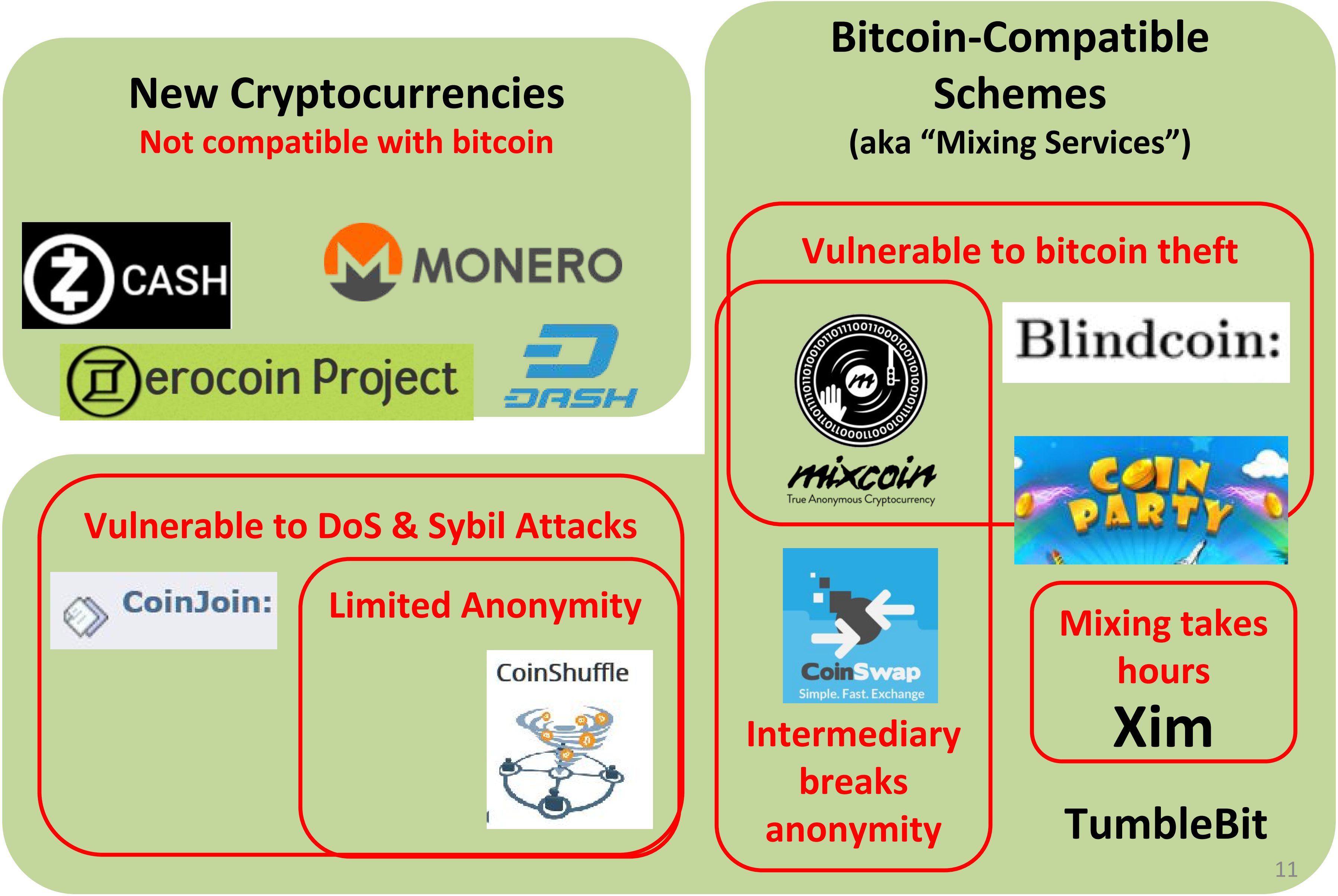
```
558dda4ede9af2da1f433514a28910561e7c9c797676e2953fff3ee46ecf3832        (Fee: 0.00013411 BTC - Size: 448 bytes) 2016-08-10 18:25:55

ee47a9e374518c294af2fae93790c9ba66                                                                              2016-08-10 18:25:54
3ChpcFbdMet9TD6z99QGqU6QJ5ZQ                                                                                      0.00026889 BTC
                                                                                                                  0.00026889 BTC

7052428ddebf61174162af76545f505d83c92cdb8b00ae0417d65bd25dd95106        (Fee: 0.00013411 BTC - Size: 448 bytes) 2016-08-10 18:25:54
393bZgKFqEGXUscbDMWe4fbn2xbNu9D42D (0.000403 BTC - Output)    1DELLrLtVCdrvW5ZwCXy6oW9WvHRBLaugE - (Spent)       0.00026889 BTC
                                                                                                                  0.00026889 BTC

d43232fac50be1ff7c8dd70fbdb12bcddf59ccea9387bc504b8a75fc14b08e0f        (Fee: 0.00013411 BTC - Size: 448 bytes) 2016-08-10 18:25:54
3HDdyjpBVd2CTxesqLN9qAgpUJQcsRA6jB (0.000403 BTC - Output)    1DELLrLtVCdrvW5ZwCXy6oW9WvHRBLaugE - (Spent)       0.00026889 BTC
                                                                                                                  0.00026889 BTC
```

You can see the transactions on the mainnet blockchain.
TXIDs avaliable in our paper.

Our implementation is Performant (per TumbleBit payment):

- 326 KB of Bandwidth,
- Puzzle-Solver takes ~0.4 seconds to compute
- Total time depends on network latency:
  No latency ~0.6 seconds.
  Boston to Tokyo ~6 seconds(clear) and ~11 seconds(both parties user TOR)

Related Work



New Cryptocurrencies

Not compatible with bitcoin

Z CASH

MONERO

Zerocoin Project

DASH

Bitcoin-Compatible Schemes

(aka "Mixing Services")

Vulnerable to bitcoin theft

mixcoin

True Anonymous Cryptocurrency

Blindcoin:

COIN PARTY

CoinSwap

Simple. Fast. Exchange

Intermediary breaks anonymity

Mixing takes hours

Xim

TumbleBit

Vulnerable to DoS & Sybil Attacks

CoinJoin:

Limited Anonymity

CoinShuffle

Conclusion

TumbleBit provides:
private untrusted scalable payments via today's Bitcoin:

1. **Private:** Unlinkable or k-anonymous payments
2. **Trustless:** Tumbler can not steal or link payments.
3. **Scalable (payment hub):** scales Bitcoin's transaction velocity and volume.

We have running code (for TumbleBit classic tumbler):

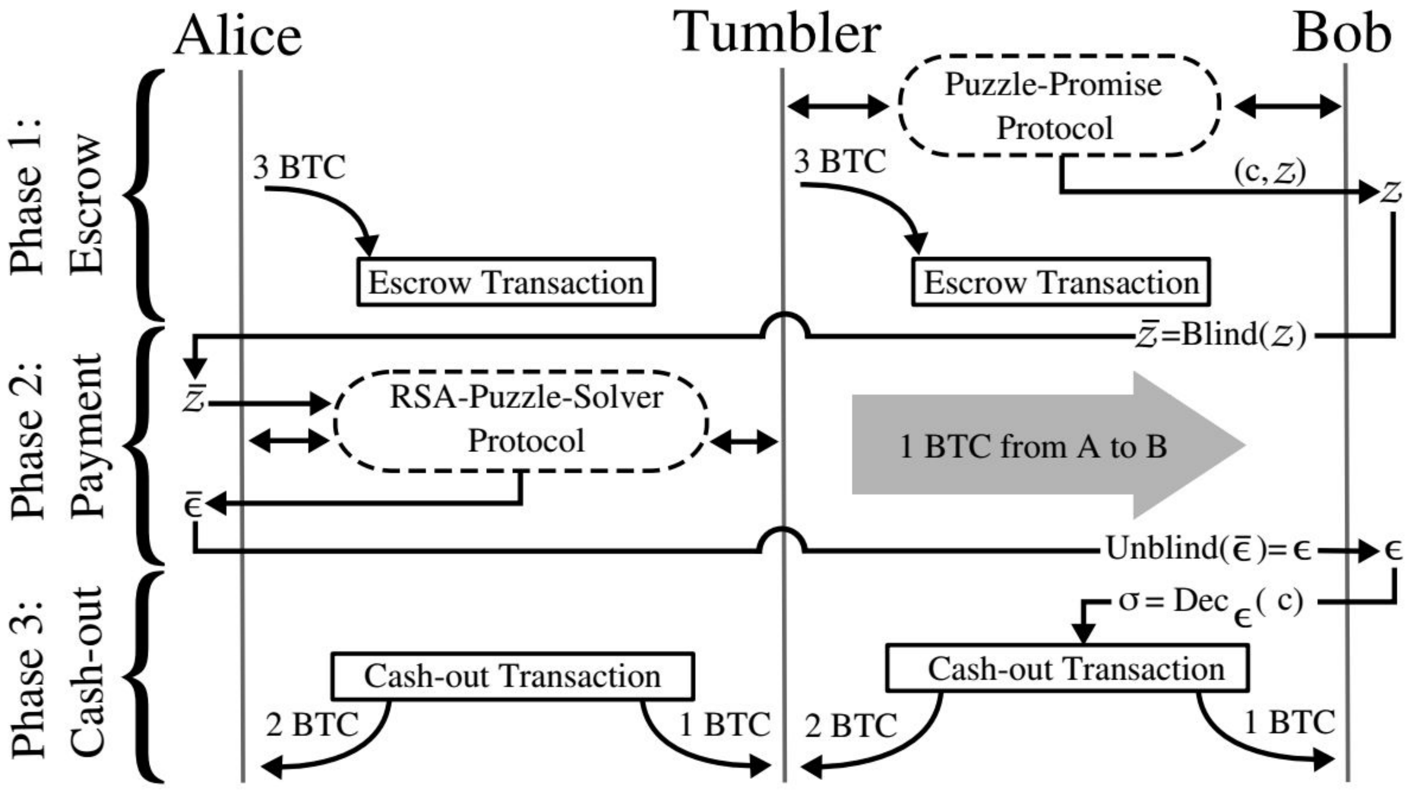
- Our code runs on Bitcoin's mainnet blockchain.
- We have published our code on github..
- ...and we working to improve it and make TumbleBit easy and safe to use.

We are hiring a full time engineer (Boston), email me if interested.

Questions?

Source code + roadmap: https://github.com/BUSEC/TumbleBit

Paper: https://eprint.iacr.org/2016/575.pdf



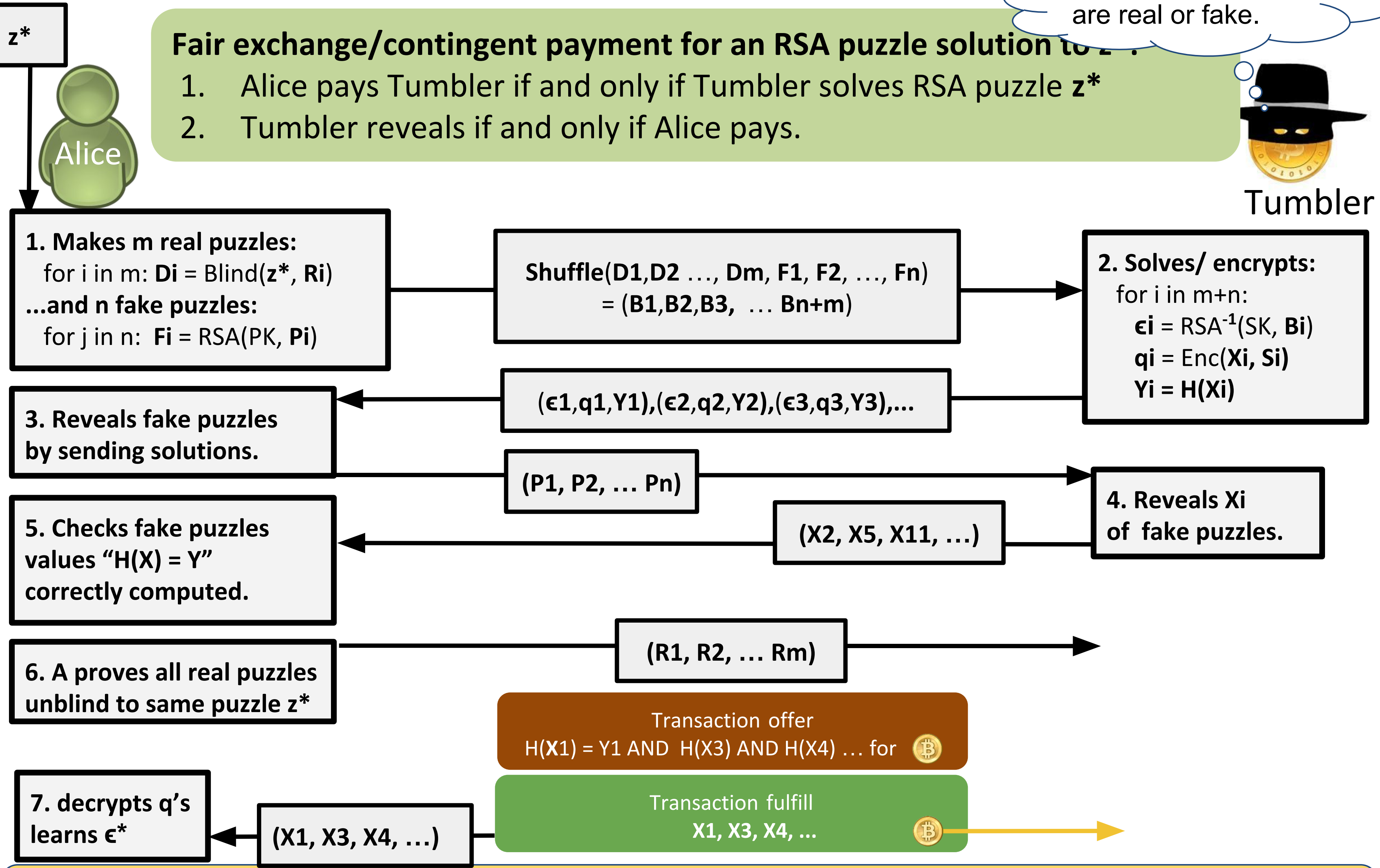
Ask questions on twitter: @Ethan_Heilman

TumbleBit: Puzzle-Solver-Pro

I can't tell which B's are real or fake.

Fair exchange/contingent payment for an RSA puzzle solution to z

1. Alice pays Tumbler if and only if Tumbler solves RSA puzzle **z***
2. Tumbler reveals if and only if Alice pays.



1. Makes m real puzzles:
for i in m: **Di** = Blind(**z***, **Ri**)
...and n fake puzzles:
for j in n: **Fi** = RSA(PK, **Pi**)

Shuffle(**D1**,**D2** …, **Dm**, **F1**, **F2**, …, **Fn**)
= (**B1**,**B2**,**B3**, … **Bn+m**)

2. Solves/ encrypts:
for i in m+n:
ϵi = RSA^{-1}(SK, **Bi**)
qi = Enc(**Xi, Si**)
Yi = H(Xi)

(ϵ1,q1,Y1),(ϵ2,q2,Y2),(ϵ3,q3,Y3),...

3. Reveals fake puzzles by sending solutions.

(P1, P2, … Pn)

4. Reveals Xi of fake puzzles.

(X2, X5, X11, …)

5. Checks fake puzzles values "H(X) = Y" correctly computed.

(R1, R2, … Rm)

6. A proves all real puzzles unblind to same puzzle z*

Transaction offer
H(**X**1) = Y1 AND H(X3) AND H(X4) … for

Transaction fulfill
X1, X3, X4, ...

7. decrypts q's learns ϵ*

(X1, X3, X4, …)

If Tumbler computes any (**qi,ϵi,Yi**) of the real puzzles correctly Alice learns **ϵ***,
thus to cheat Alice, Tumbler must corrupt all the real and none of the fake puzzles.

TumbleBit: Puzzle-Promise-Protocol

At the end of this protocol: Bob should be convinced that for a (**z, c**):

1. The ciphertext **c** decrypts to **σ** under a key **ϵ** i.e Dec(**ϵ,c) = σ**
2. **AND** the key **ϵ** is the solution to the RSA-puzzle **z.**

The protocol should never: allow Bob to learn a valid **σ** (without paying).

Tumbler

1. B sends: a mix of hashes of valid and invalid claim transactions.

B = H(T1),H(T2🚫),H(T3🚫),H(T4),H(T5🚫),H(T6)

2. T Signs & Encrypts σ:
for **Bi** in **B**:
σi = Si
zi = RS

This is why the protocol is hard,
otherwise Tumbler could convince Bob

s
s.

Probability(Tumbler successfully cheats) = (m+n choose m) = ~1/(2^{80})

m = # of valid transactions = 15

n = # of invalid transactions = 285

4. T Reveals: ϵi for invalid transactions.

ϵ2,**ϵ3**,**ϵ5**

5. B checks: invalid transactions **σi** are correctly computed.

6. Bob and Tumbler run "quotient protocol" ensuring that:
if Bob learns **ϵ1,** Bob can use that knowledge to learn **ϵ4,ϵ6.**
(**ϵ4/ϵ1** mod N**, ϵ6/ϵ4** mod N)

If Tumbler computes any (**ϵi,σi**) of the valid transactions correctly Bob learns a **σ**/gets paid,
thus to cheat Bob, Tumbler must all corrupt all the valid and none of the invalid transactions.

TumbleBit: Future Roadmap

People want TumbleBIt...

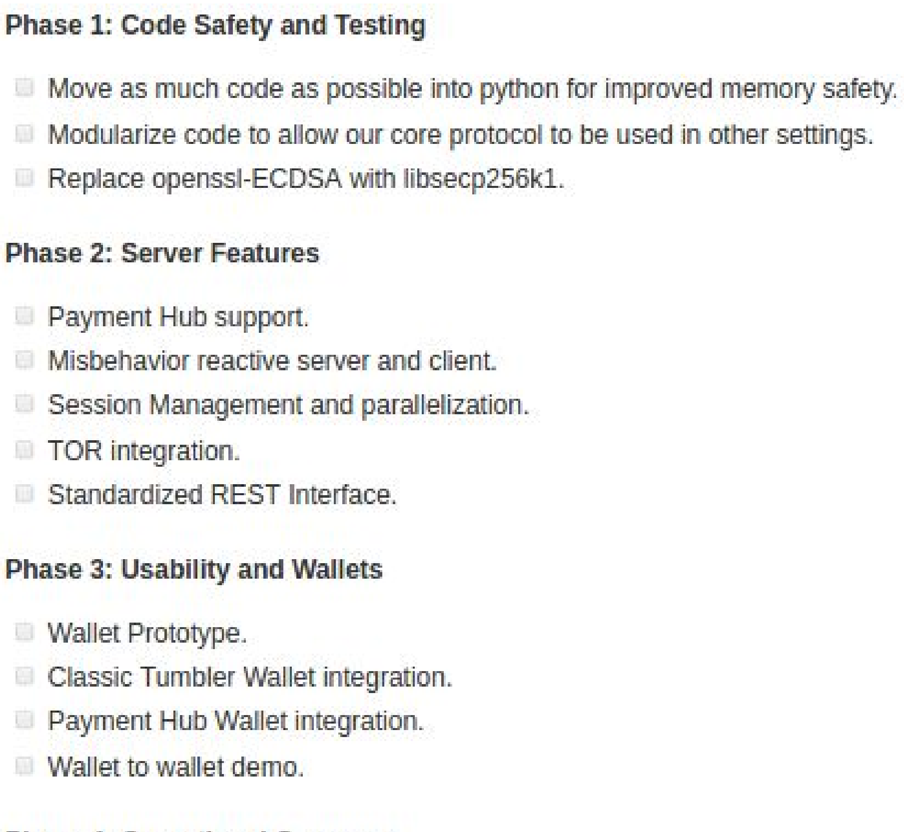
Ethan ✨ Heilman @Ethan_Heilman · Aug 29

4,943

New version of TumbleBit rewritten to focus on anonymity of #Bitcoin payment hubs/micropayment channels: eprint.iacr.org/2016/575 #privacy

C++ ★ 42

but to get TumbleBit into the hands of everyday users we need to build ...secure, safe, and usable software.



Phase 1: Code Safety and Testing

- ☐ Move as much code as possible into python for improved memory safety.
- ☐ Modularize code to allow our core protocol to be used in other settings.
- ☐ Replace openssl-ECDSA with libsecp256k1.

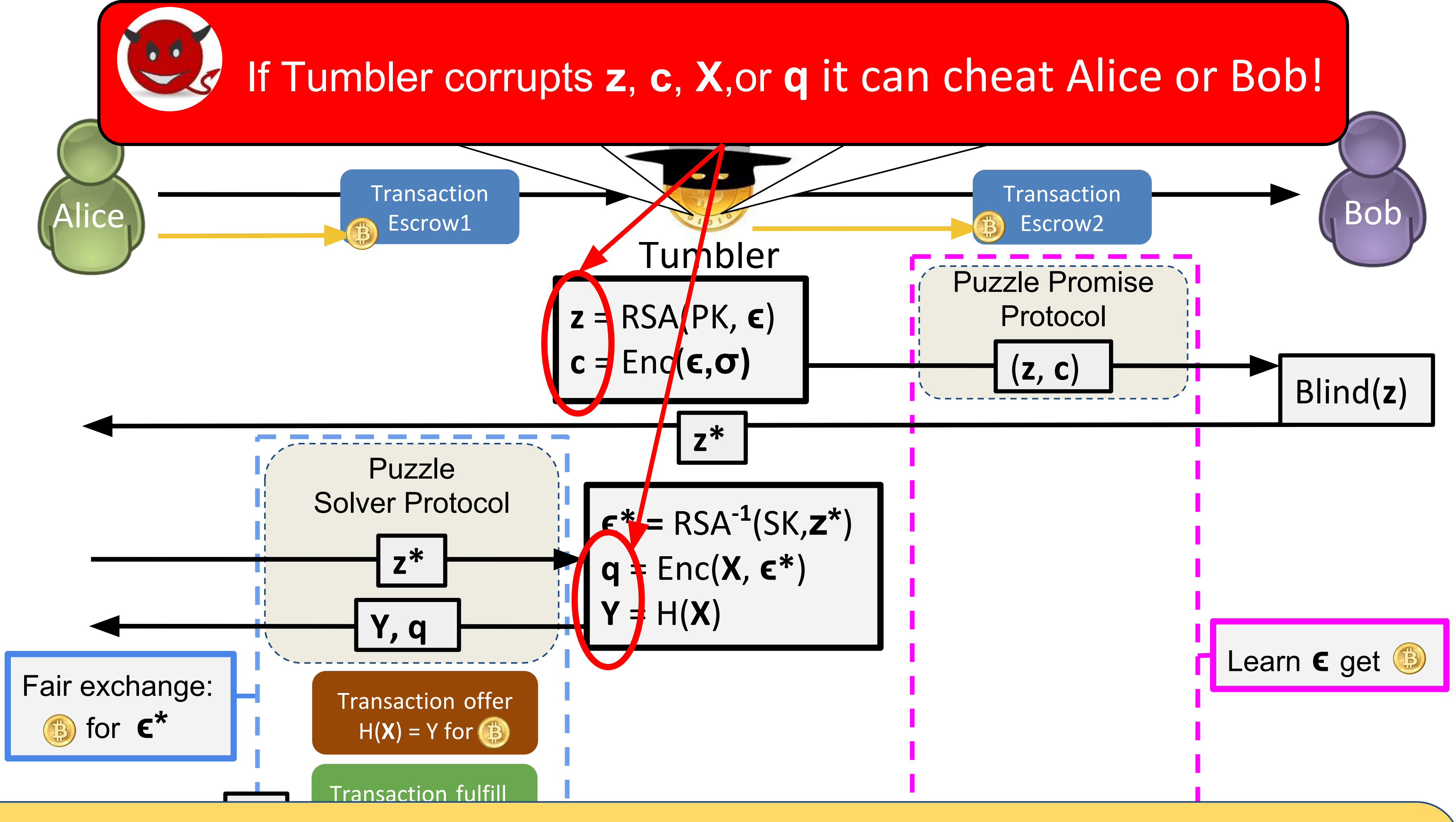
Phase 2: Server Features

- ☐ Payment Hub support.
- ☐ Misbehavior reactive server and client.
- ☐ Session Management and parallelization.
- ☐ TOR integration.
- ☐ Standardized REST Interface.

Phase 3: Usability and Wallets

- ☐ Wallet Prototype.
- ☐ Classic Tumbler Wallet integration.
- ☐ Payment Hub Wallet integration.
- ☐ Wallet to wallet demo.

TumbleBit: Protocol Overview



If Tumbler corrupts **z**, **c**, **X**,or **q** it can cheat Alice or Bob!

Alice

Transaction Escrow1

Tumbler

Transaction Escrow2

Bob

$\mathbf{z} = \mathrm{RSA}(PK, \boldsymbol{\epsilon})$

$\mathbf{c} = \mathrm{Enc}(\boldsymbol{\epsilon},\boldsymbol{\sigma})$

Puzzle Promise Protocol

$(\mathbf{z}, \mathbf{c})$

Blind($\mathbf{z}$)

$\mathbf{z}^*$

Puzzle Solver Protocol

$\mathbf{z}^*$

$\mathbf{Y}, \mathbf{q}$

$\boldsymbol{\epsilon}^* = \mathrm{RSA}^{-1}(SK,\mathbf{z}^*)$

$\mathbf{q} = \mathrm{Enc}(\mathbf{X}, \boldsymbol{\epsilon}^*)$

$\mathbf{Y} = \mathrm{H}(\mathbf{X})$

Learn $\boldsymbol{\epsilon}$ get ₿

Fair exchange: ₿ for $\boldsymbol{\epsilon}^*$

Transaction offer H(**X**) = Y for ₿

Transaction fulfill

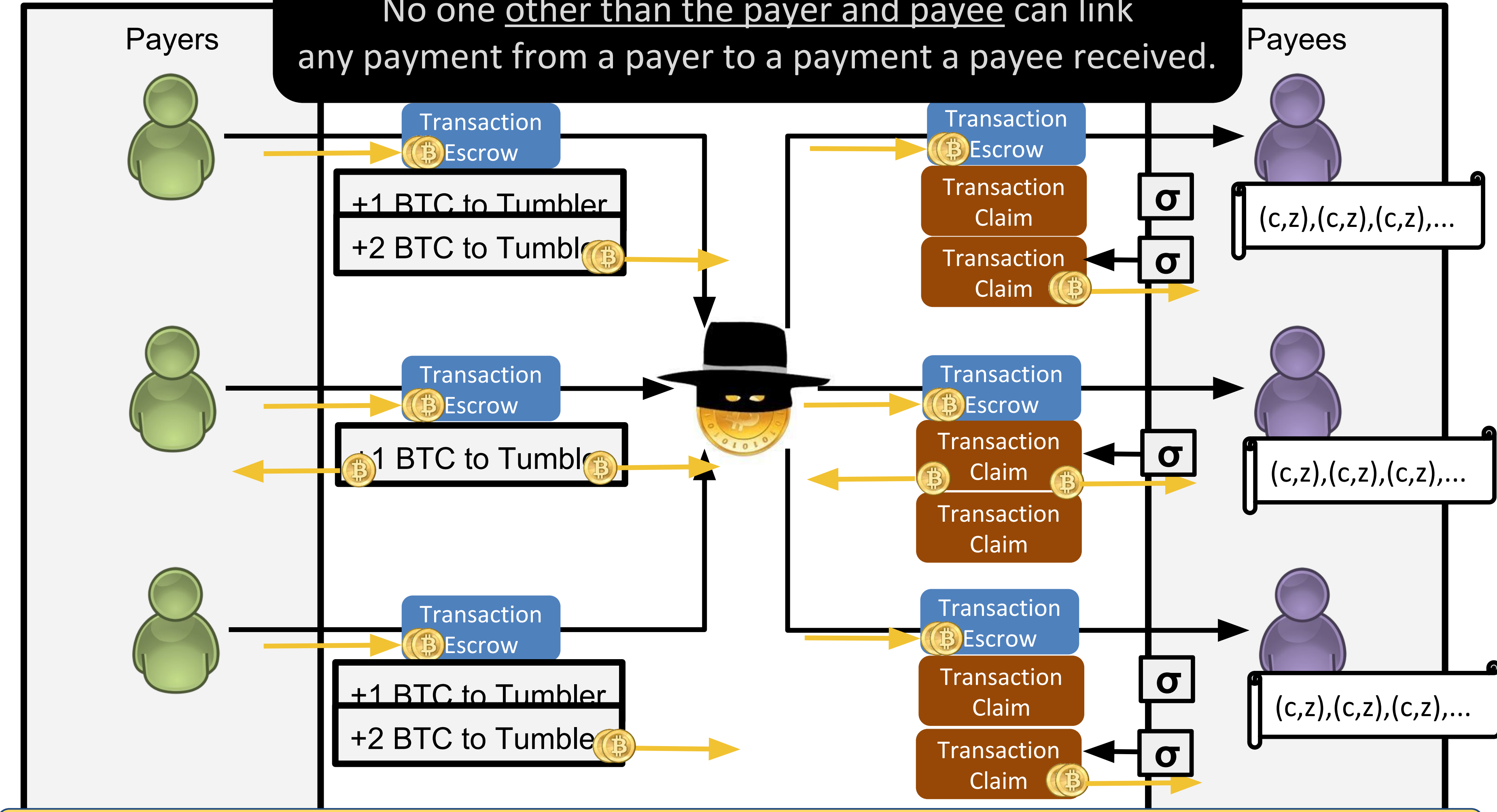
TumbleBit prevents this via two protocols:

Puzzle-Solver-Protocol:
Tumbler convinces Alice the preimage **X** where Hash(**X**) = **Y** will allow her to learn **ϵ***.

Puzzle-Promise-Protocol:
Tumbler convinces Bob that the solution to RSA puzzle **z** is a value **ϵ** which allows him learn **σ**.

TumbleBit: Privacy

Payments are unlinkable:
No one other than the payer and payee can link
any payment from a payer to a payment a payee received.

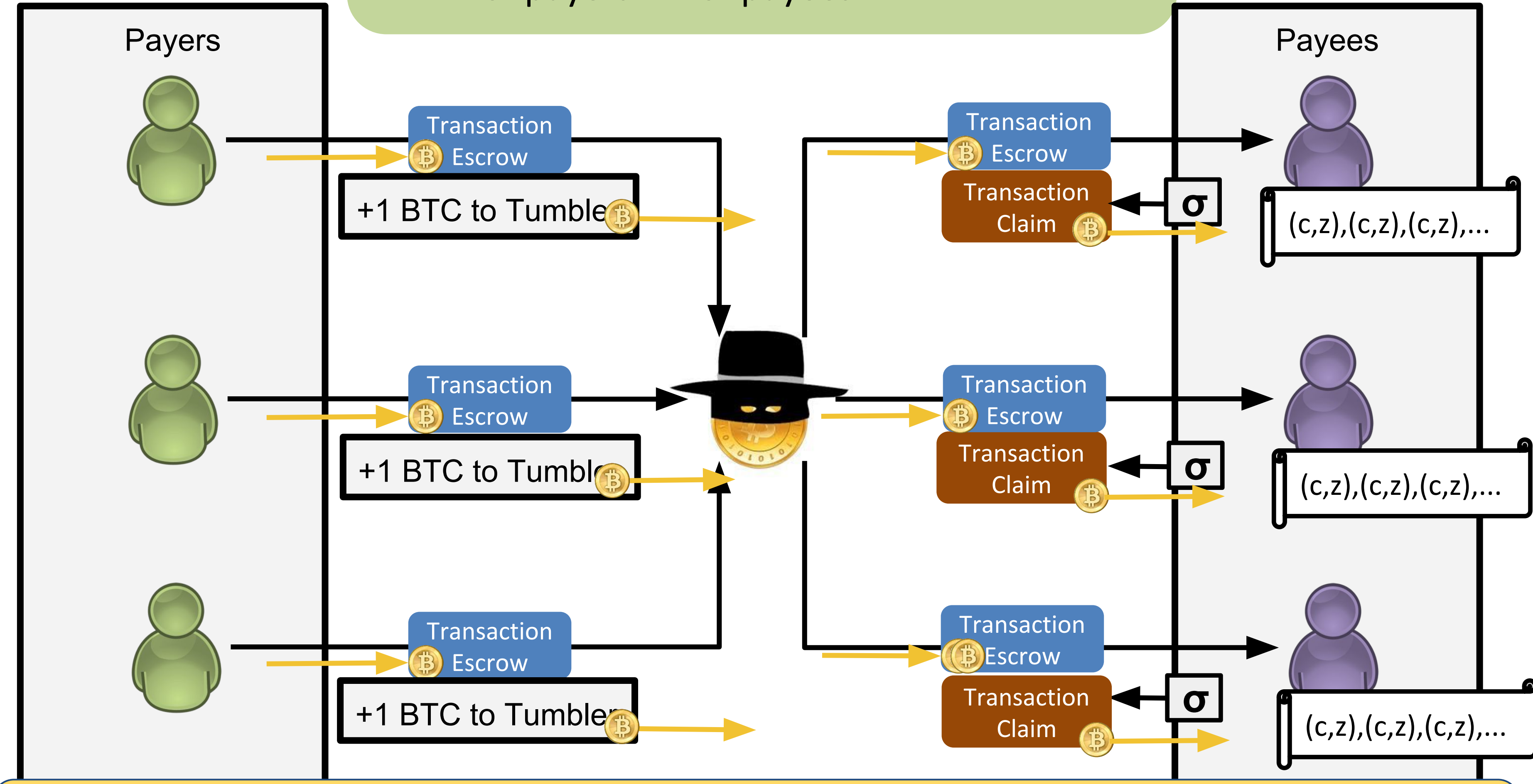


Tumbler learns: (1) payer & time of payment, (2) # of payments each payee received.

TumbleBit: Classic Tumbler

To run TumbleBit as a Classic Bitcoin Tumbler:

- Each payer just makes one payment.
- Each payee accepts only one payment.
- # of payers = # of payees.



Provides k-anonymity:

Where k = # of payers = # of payee.

TumbleBit: Implementation

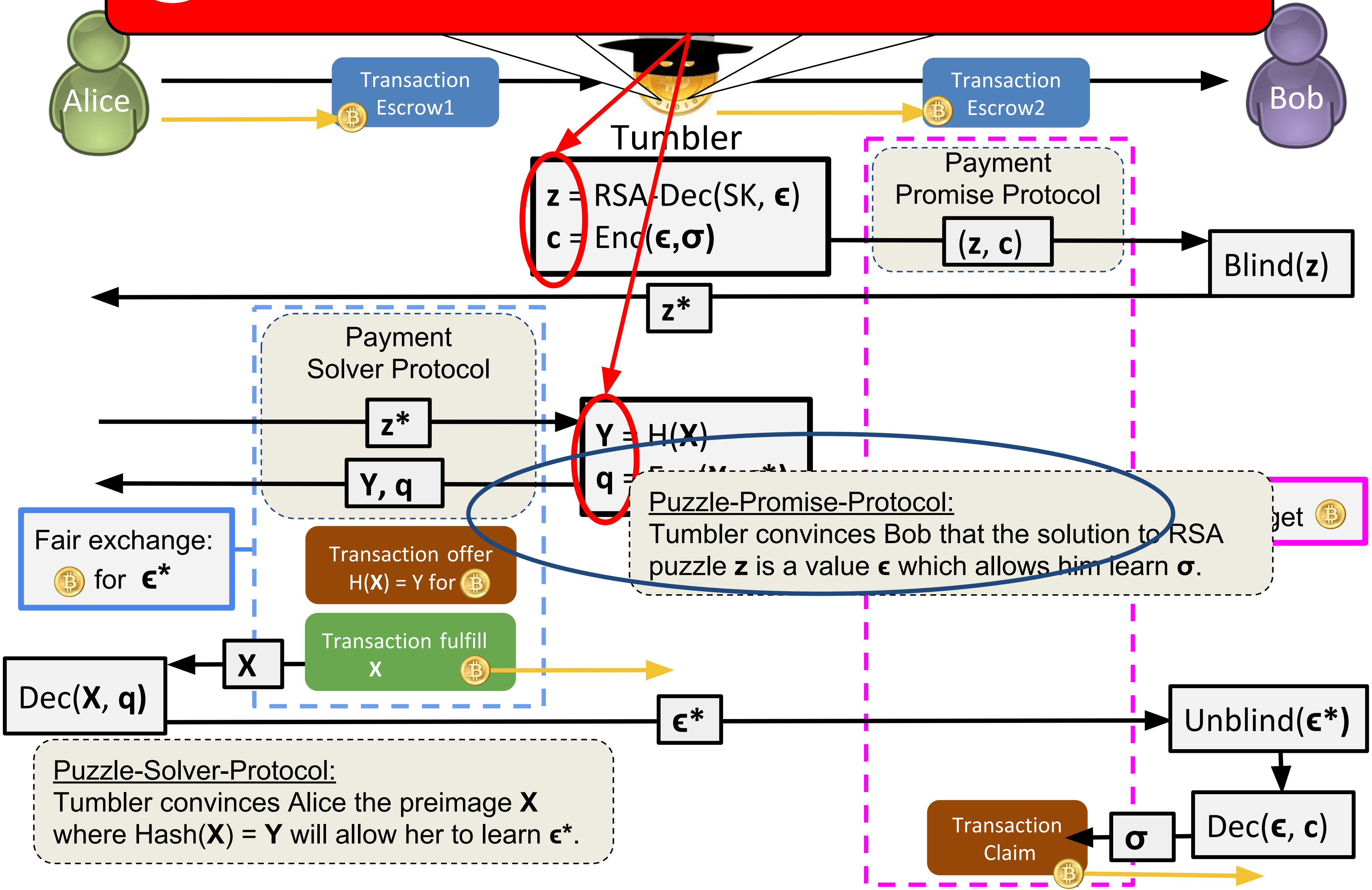
1. **We wrote a proof-of-concept implementation:**
    - Source code is available on Github.
    - We are working to improve it to make it user ready.

2. **We “tumbled” 800 addresses to 800 addresses:**
    - In our paper we provide links to runs on Bitcoin’s blockchain (mainnet).

3. **Our implementation is Performant:**
    - 326 KB of Bandwidth.
    - Computation time 0.3 - 0.6 seconds.
    - Total time depends on network latency:
      No latency ~0.6 seconds.
      Boston to NYC ~1.6 seconds.
      Boston to Tokyo ~ 4.18 seconds.

TumbleBit: Protocol Overview

If Tumbler corrupts **z**, **c**, **X**,or **q** it can cheat Alice or Bob!

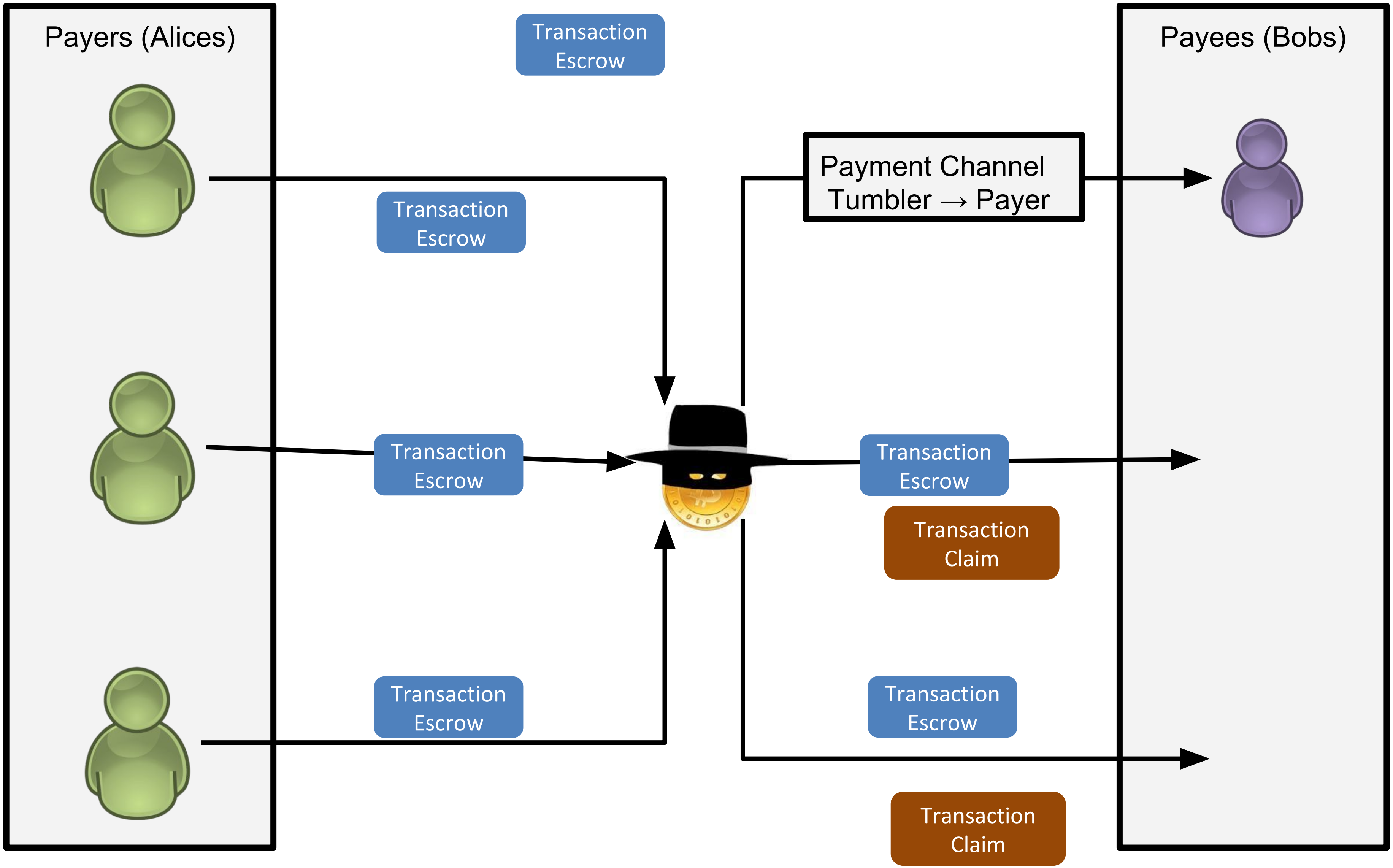


Fair exchange: for **ϵ***

Puzzle-Promise-Protocol:
Tumbler convinces Bob that the solution to RSA puzzle **z** is a value **ϵ** which allows him learn **σ**.

Puzzle-Solver-Protocol:
Tumbler convinces Alice the preimage **X** where Hash(**X**) = **Y** will allow her to learn **ϵ***.

Payment Hubs: Privacy

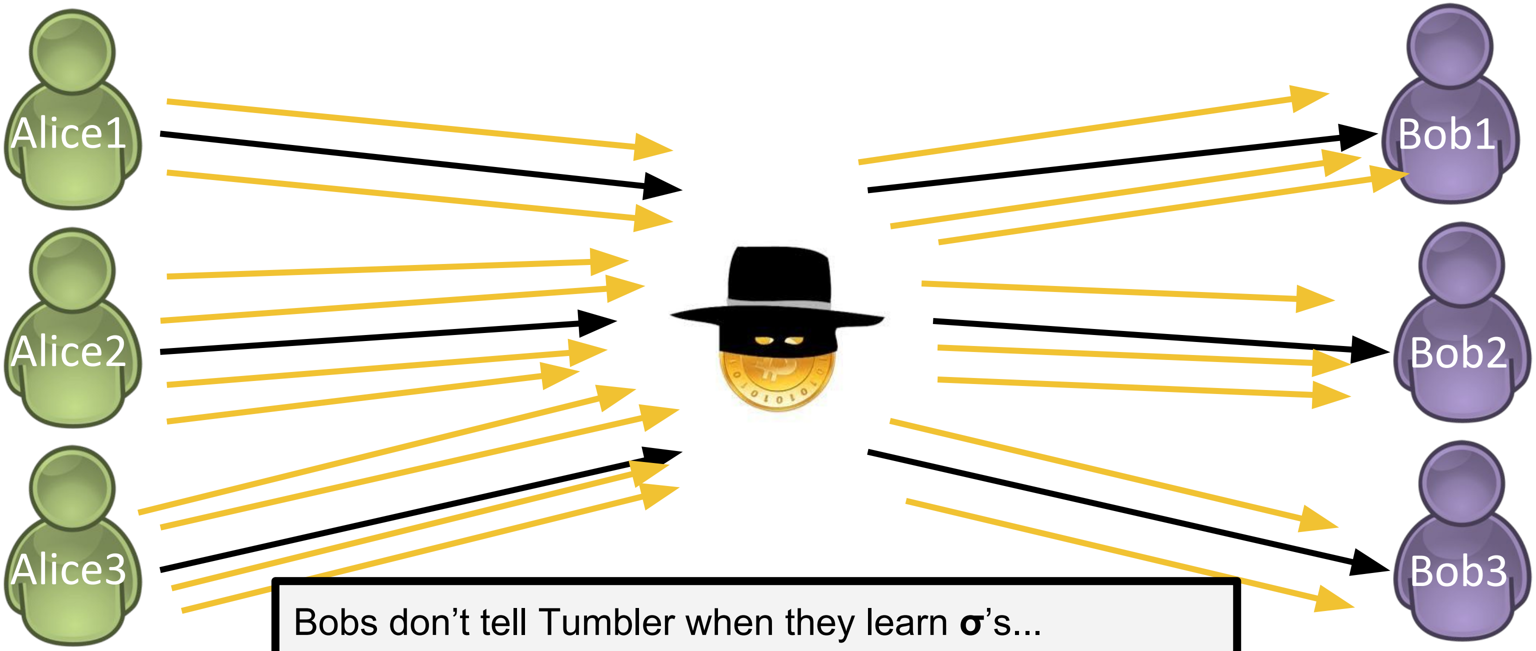


TumbleBit: Phases and Privacy

1. Escrow Phase: All payment channels setup.

2. Payments Phase (~1 month): Alices make many payments to Bobs.

3. Cashout Phase: Bobs and Alices close their payment channels.



Bobs don't tell Tumbler when they learn **σ**'s...
...thus, Tumbler only see's Alice to Tumbler payments.
This prevents Tumbler from performing a timing attack..

Tumbler learns two sets of things:
1. that an Alice paid an unknown party at time t,
2. during the payment phase the total # of payments each Bob received..

Puzzle-Promise-Protocol

At the end of this protocol: Bob should be convinced that for a (**z, c**):

1. The ciphertext **c** decrypts to **σ** under a key **ϵ** i.e Dec(**ϵ,c**) **= σ**
2. **AND** the key **ϵ** is the solution to the RSA-puzzle **z** i.e **z** = ϵ^{pk} mod N

The protocol should never: allow Bob to learn a valid **σ** (without paying Tumbler).

T-PK,
T-SK.

Tumbler

Why preve
σ allows B
Thus, Alice

1. Bob creates and randomly permutes:
m - valid transactions (reals) ■ ■ ■
n - invalid transactions (fakes) 🚫🚫🚫
B = A randomly permuted list of the real and fake transactions hashes.

TumbleBit sets (m = 42, n = 42):
Prob.(Tumbler successfully cheats) = 2^{-80}

2. Tumbler signs each B
For all Bi in B: **zi** = **ϵi** mod N, **σi**=Sig(T-SK, Bi), **ci** = Enc(**ϵi,σi**)

(z1,c1),(z2,c2),(z3,c3),(z4,c4),(z5,c5),(z6,c6)

3. Bob reveals which of the B's are fake.

B = {R,F,F,R,F,R}

4. Tumbler confirms fakes are really fakes. Reveals fake **ϵi**'s.

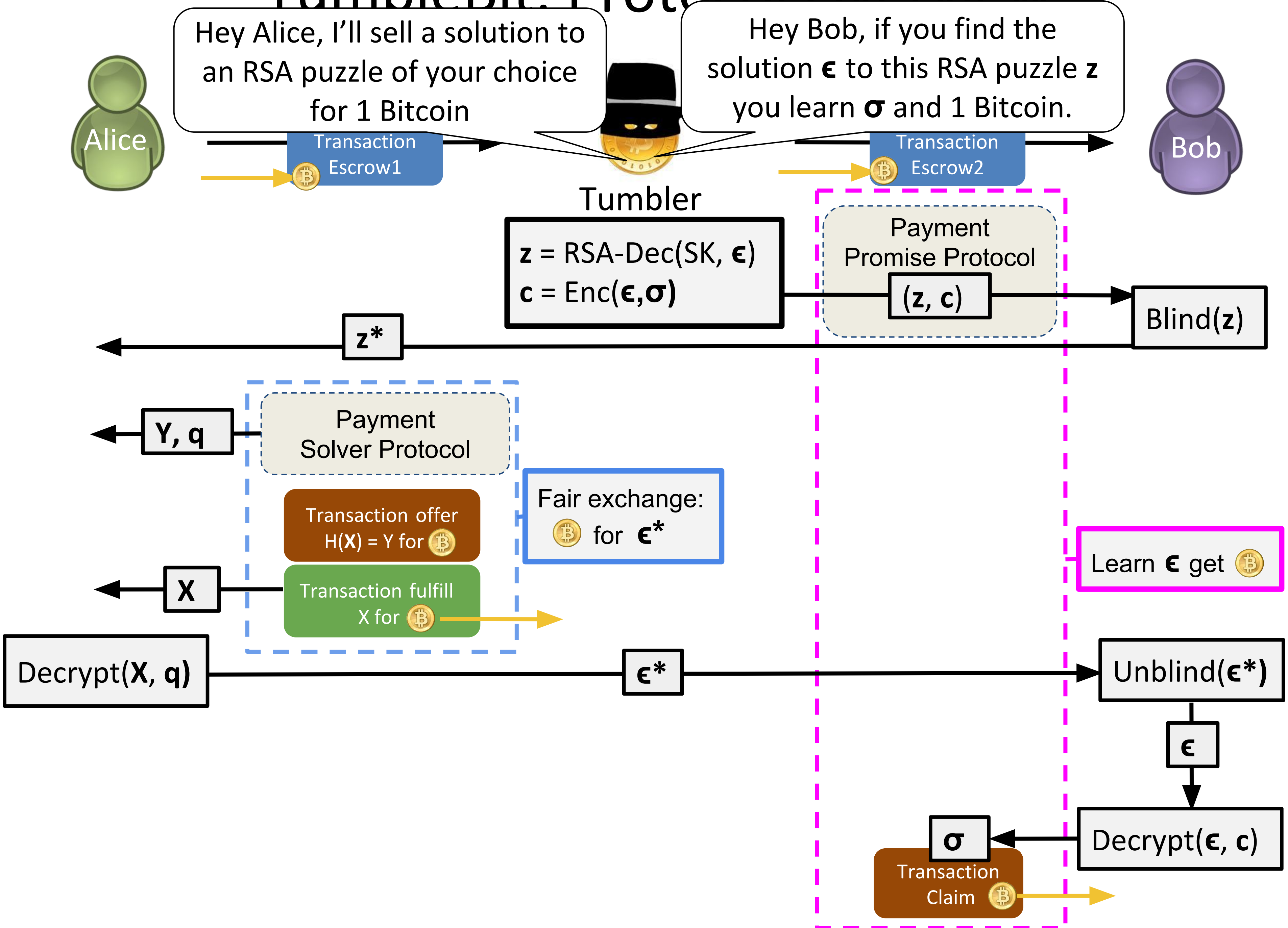
ϵ2,ϵ3,ϵ5

5. Bob checks that Tumbler computed fakes honestly.

6. Bob and Tumbler run "the quotient protocol" ensuring that:
if Bob learns **ϵ1,** Bob can use that knowledge to learn **ϵ4,ϵ6.**

If Tumbler computes any (**ϵi,σi**) of the reals correctly then Bob learns a **σ**/gets paid,
Thus, to cheat Bob, Tumbler must all corrupt all the reals and none of the fakes.
Prob(Tumbler successfully cheats) = 1/(m+n choose m)

TumbleBit: Protocol Overview



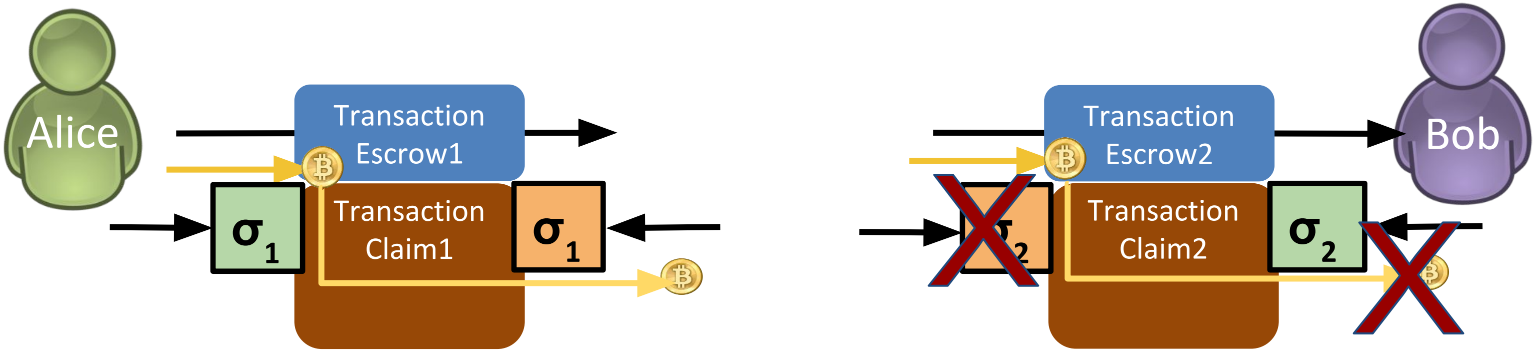
Hey Alice, I'll sell a solution to an RSA puzzle of your choice for 1 Bitcoin

Hey Bob, if you find the solution **ϵ** to this RSA puzzle **z** you learn **σ** and 1 Bitcoin.

TumbleBit: Protocol Overview

Solution $\boldsymbol{\in}$ RSA-puzzle $\mathbf{z} \rightarrow \boldsymbol{\sigma}$

TumbleBit Protocol



Alice signs Claim1

Payment Hub and Bob could sign and post both claim transactions, paying 1 Bitcoin from Alice to Bob via the Payment Hub.

...But what if the hub is malicious,

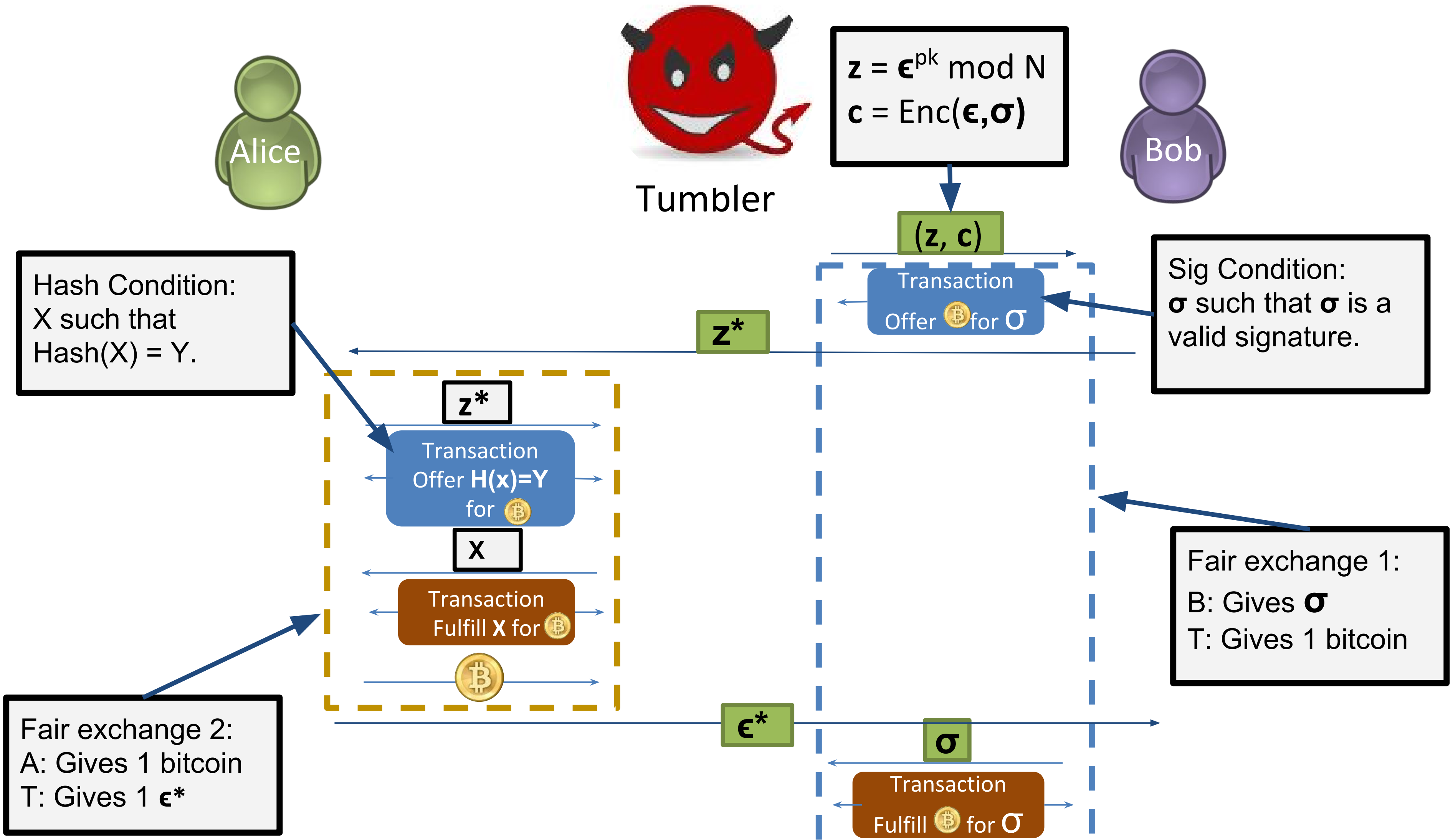
Atomicity: If Claim1 and Claim2 happen atomically then theft is prevented.

Hash locks provide this property.

Puzzle-Solver-Protocol:
Tumbler convinces Alice the preimage X where Hash(X) = Y will allow her to learn ϵ*.

Puzzle-Promise-Protocol:
Tumbler convinces Bob that the solution to RSA puzzle **z** is a value **ϵ** which allows him learn **σ** and thereby claim 1 Bitcoin.

But what if the Tumbler is malicious and cheats Alice and Bob?



$z = \epsilon^{pk} \bmod N$
$c = Enc(\epsilon, \sigma)$

Hash Condition:
X such that
Hash(X) = Y.

Sig Condition:
σ such that **σ** is a valid signature.

Fair exchange 1:
B: Gives **σ**
T: Gives 1 bitcoin

Fair exchange 2:
A: Gives 1 bitcoin
T: Gives 1 **ϵ***

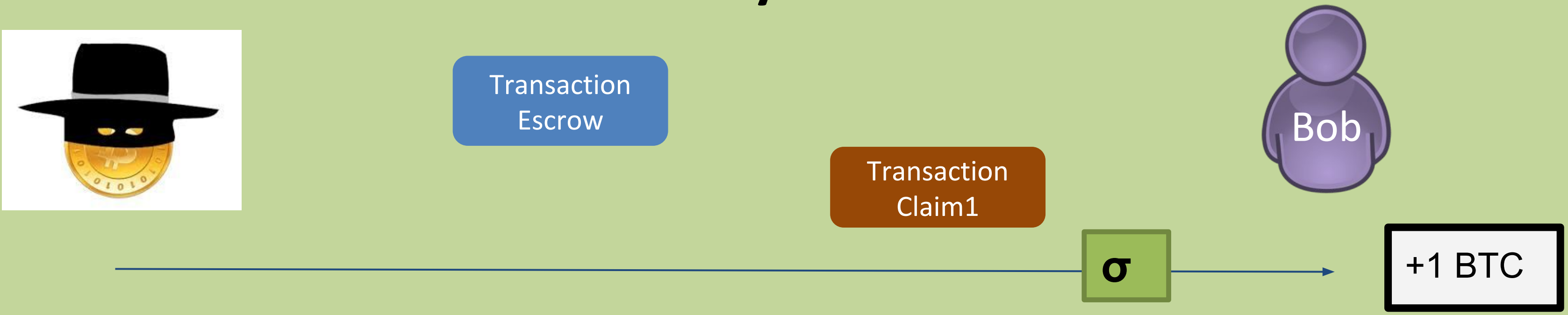
I am going to walk through the puzzle promise protocol

Alice pays Bob with RSA Puzzles

Hey Alice, I'll sell a solution to an RSA puzzle of your choice for 1 Bitcoin

Hey Bob, if you find the solution **ϵ** to this RSA puzzle **z** you get 1 Bitcoin.

Remember how Payment Channels work:



Transaction Escrow

Transaction Claim1

σ

+1 BTC

Tumbler can encrypt **σ** under an RSA-puzzle

z = **ϵ**^{pk} mod N
c = Enc(**ϵ**,**σ**)

(c, z)

If Bob learns the solution **ϵ** to **z**
Bob can decrypt **c** to **σ** and get 1 BTC.

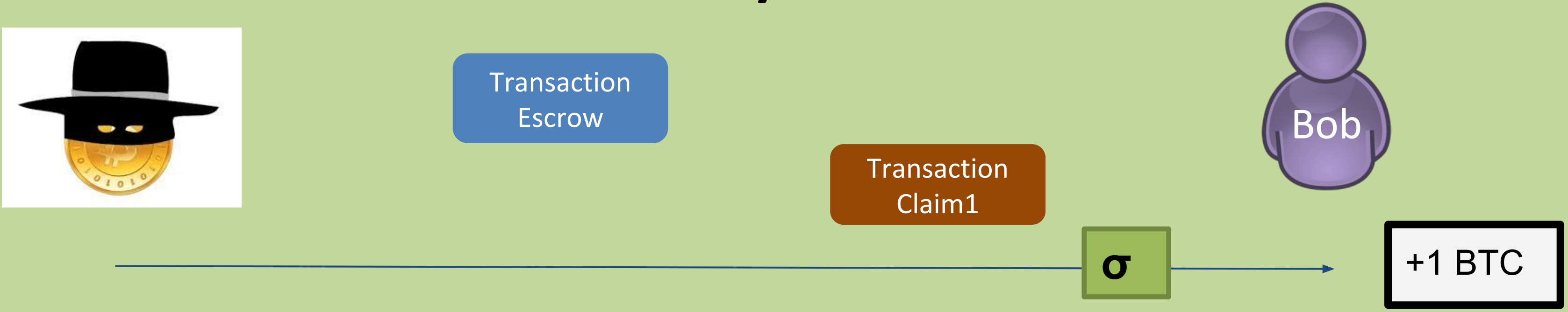
+1 BTC

Alice pays Bob with RSA Puzzles

Hey Alice, I'll sell a solution to an RSA puzzle of your choice for 1 Bitcoin

Hey Bob, if you find the solution **ϵ** to this RSA puzzle **z** you get 1 Bitcoin.

Remember how Payment Channels work:



Tumbler can encrypt σ under an RSA-puzzle

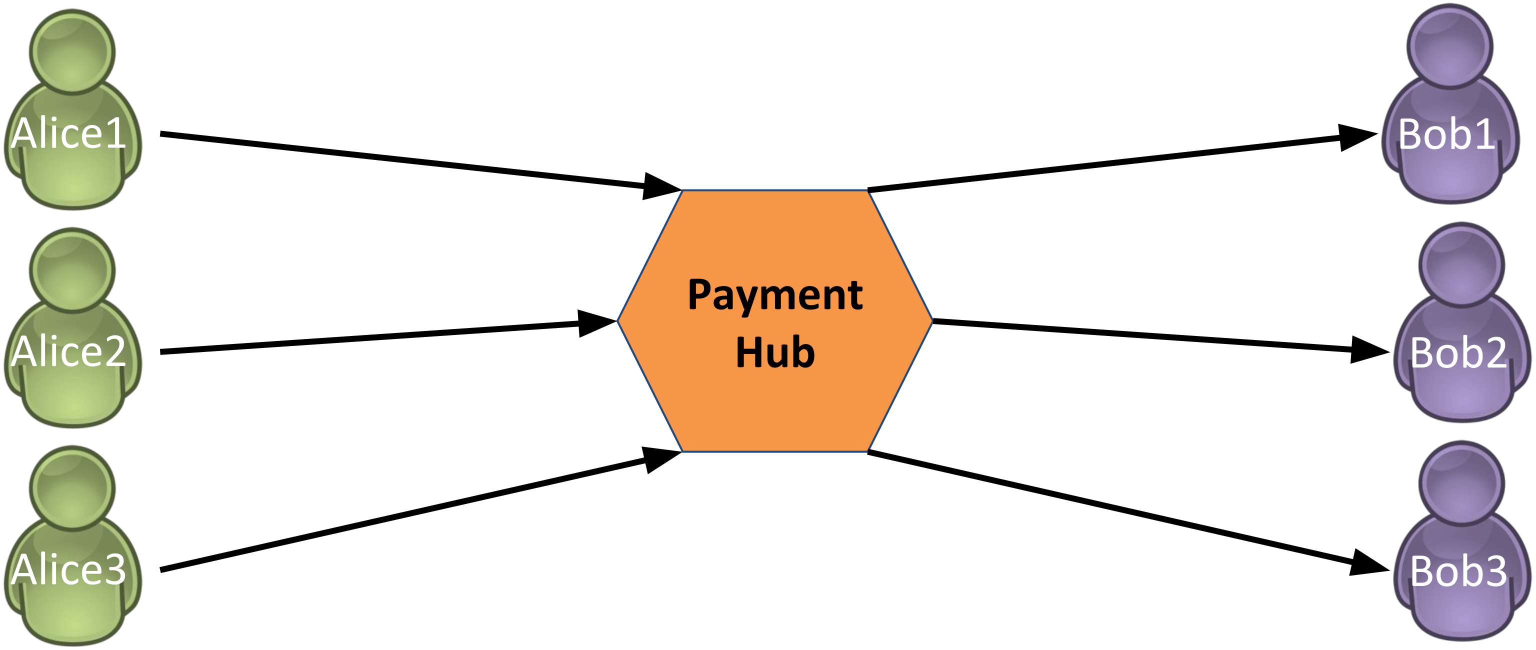
z = $\boldsymbol{\epsilon}^{pk}$ mod N
c = Enc(**ϵ**,**σ**)

(c, z)

If Bob learns the solution **ϵ** to **z**
Bob can decrypt **c** to **σ** and get 1 BTC.

+1 BTC

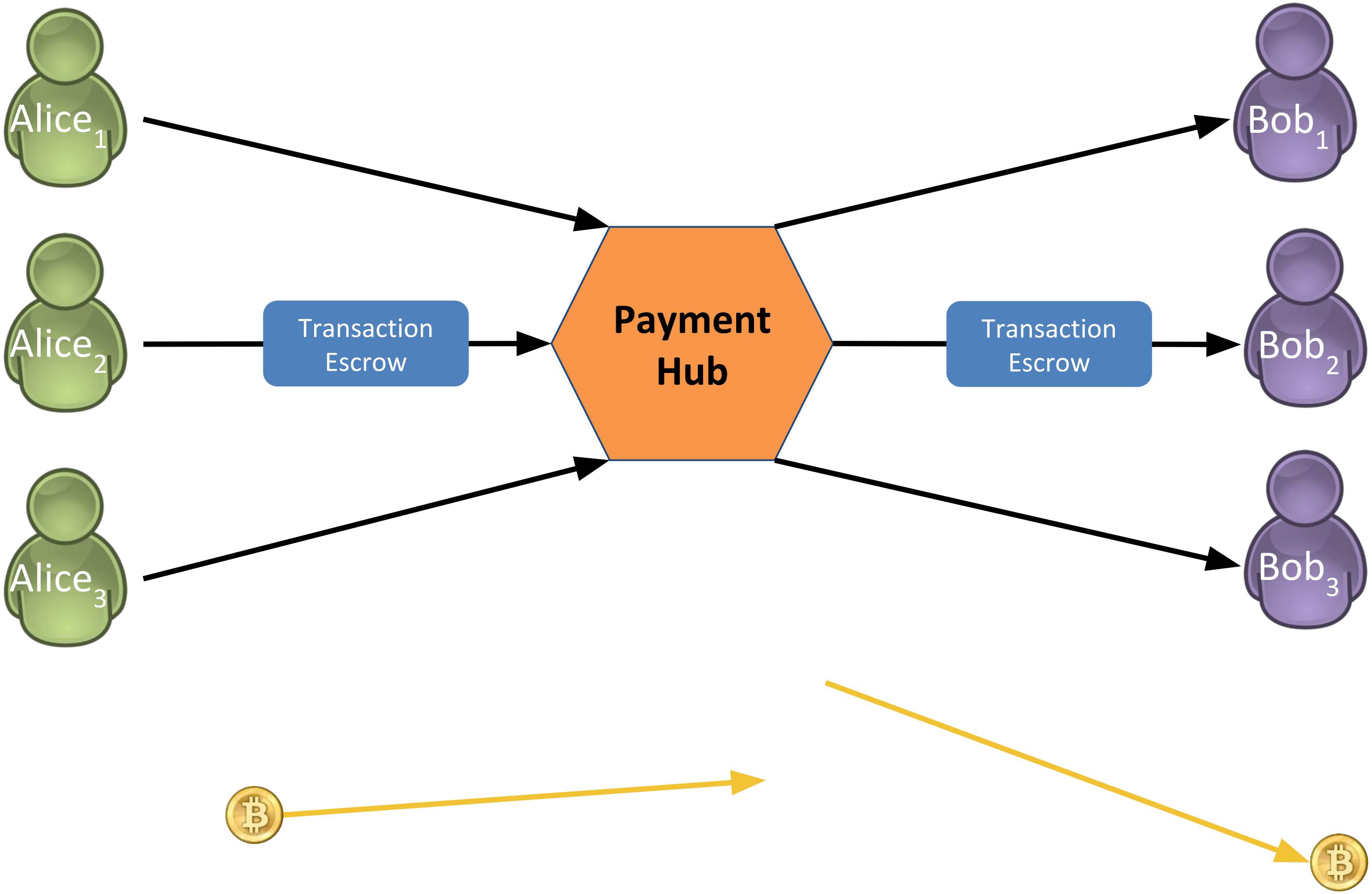
Payment Hub: Privacy



TumbleBit improves payment hubs so that
for each payment the payer can not be linked to the payees.

Payment Hub

A payment hub: routes payment channels



Introduction

Outline

- Payment hubs
  - Bitcoin transactions/payment channels
  - What are Bitcoin payment hubs?
  - Scalability benefits of payment hubs
  - Are payment hubs private?
- TumbleBit as a Payment Hub
  - RSA-blind puzzles
  - TumbleBit as an unlinkable payment hub
  - Ensuring fair-exchange (TumbleBit can’t steal)
  - Puzzle-Promise-Protocol

Motivation

Technical challenges facing Bitcoin: Privacy, Scalability

Privacy:

- Bitcoin is not anonymous
- Payment history is saved to the blockchain i.e. an eternal public record

Scaling Transaction velocity:

- Transactions are confirmed on the blockchain (avg wait time ~10 mins)
- No confirmation = double spending possible

Scaling Transaction volume:

- Bitcoin: 7 transactions/sec max throughput[1]
- Visa (average): 2000 transactions/sec[1]
- Visa (peak): 56,000 transactions/sec[1]
- Limiting factor is space in Bitcoin's blockchain

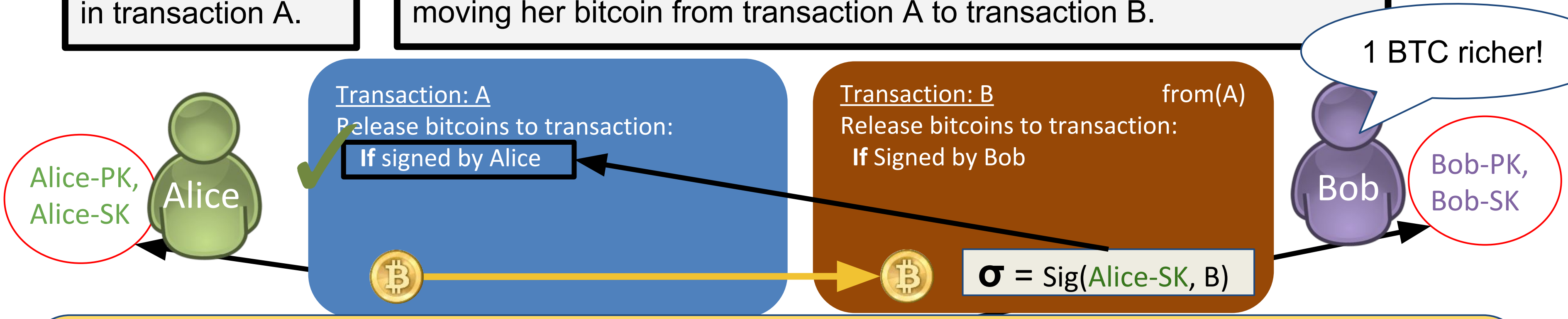
TumbleBit is designed to address these challenges by providing privacy and scalability **without** introducing trust.

[1]: 'On Scaling Decentralized Blockchains (A Position Paper)' Croman, et al.

Bitcoin Transactions

1. Alice has 1 BTC in transaction A.

2. To setup a payment of 1 BTC to Bob, Alice creates a transaction B moving her bitcoin from transaction A to transaction B.



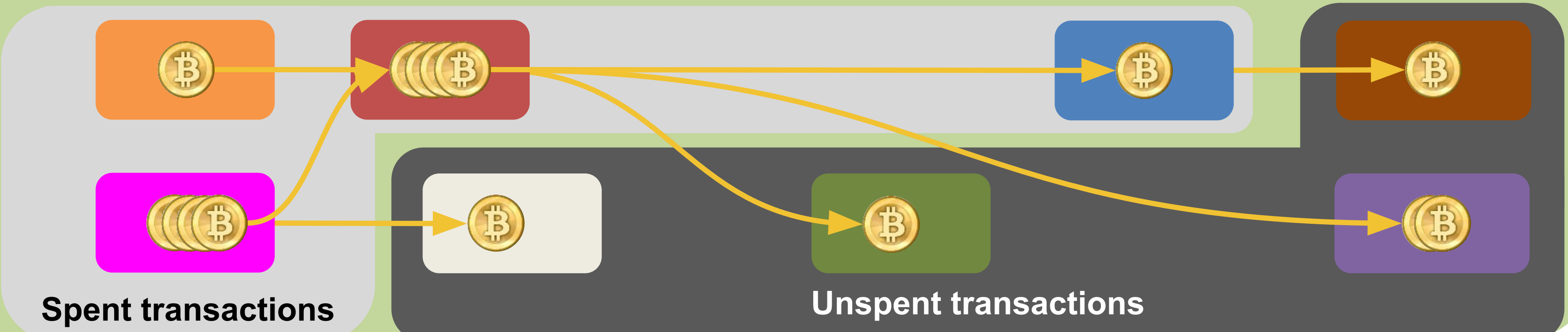
Transaction conditions ("release bitcoins to transaction if") are programmable:

- via a very limited non-turing complete language called *Script*,
- can verify multiple signatures and perform a few other operations.

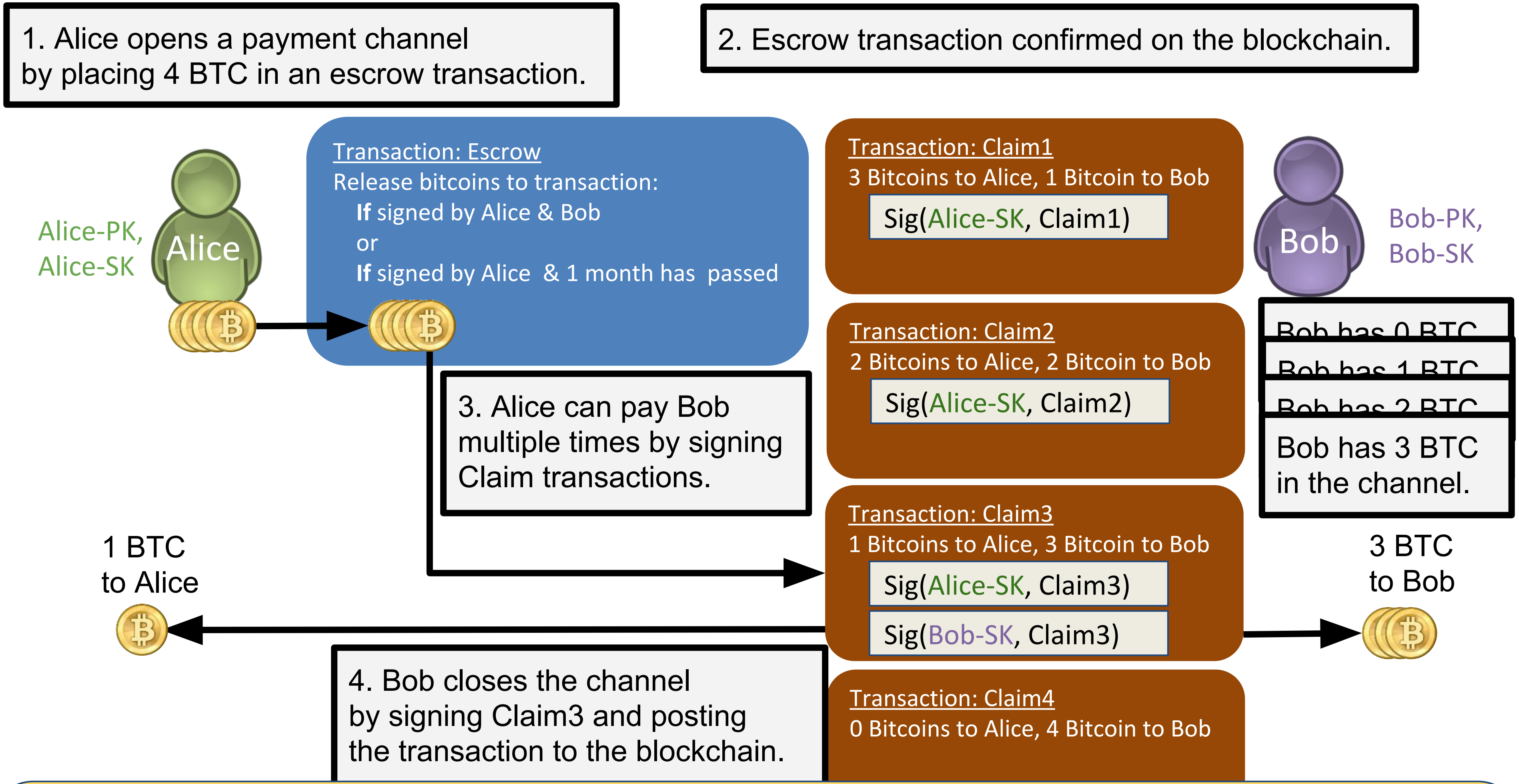
I will talk more about it later.

Payment in Bitcoin occurs by transferring bitcoins in one transaction to a new transaction...

...thus, ownership is merely holding a secret which can authorize such transfers.



Unidirectional Payment Channels



1. Alice opens a payment channel by placing 4 BTC in an escrow transaction.

2. Escrow transaction confirmed on the blockchain.

Alice-PK,
Alice-SK

Transaction: Escrow
Release bitcoins to transaction:
If signed by Alice & Bob
or
If signed by Alice & 1 month has passed

Transaction: Claim1
3 Bitcoins to Alice, 1 Bitcoin to Bob
Sig(Alice-SK, Claim1)

Transaction: Claim2
2 Bitcoins to Alice, 2 Bitcoin to Bob
Sig(Alice-SK, Claim2)

Transaction: Claim3
1 Bitcoins to Alice, 3 Bitcoin to Bob
Sig(Alice-SK, Claim3)
Sig(Bob-SK, Claim3)

Transaction: Claim4
0 Bitcoins to Alice, 4 Bitcoin to Bob

Bob-PK,
Bob-SK

Bob has 0 BTC

Bob has 1 BTC

Bob has 2 BTC

Bob has 3 BTC in the channel.

3. Alice can pay Bob multiple times by signing Claim transactions.

1 BTC
to Alice

3 BTC
to Bob

4. Bob closes the channel by signing Claim3 and posting the transaction to the blockchain.

Advantages of Payment channels

Scales Tx volume: Two transaction on the blockchain allow Alice to pay Bob many times.

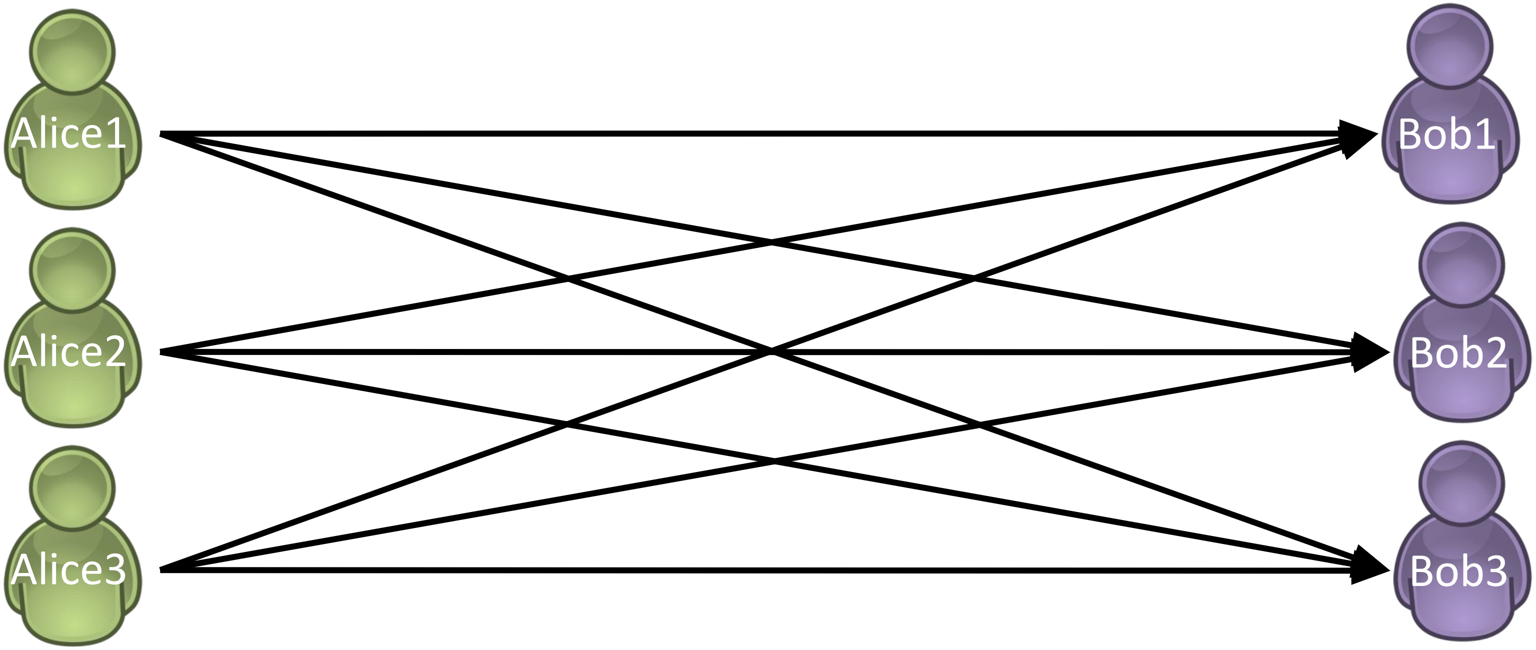
Scales Tx velocity: Risk of Double spending ~=0 so payments happen in milliseconds.

No trust required: Neither Alice nor Bob can cheat each other.
If Bob walks away Alice gets her money back after 1 month.

Unidirectional Payment Channels

Disadvantages of Payment channels:

1. To pay many different Bobs, requires many different channels.
2. Each channel setup is expensive in time (~10 minutes)
3. **...and** money (i.e. BTC sitting in escrows that can't be used).

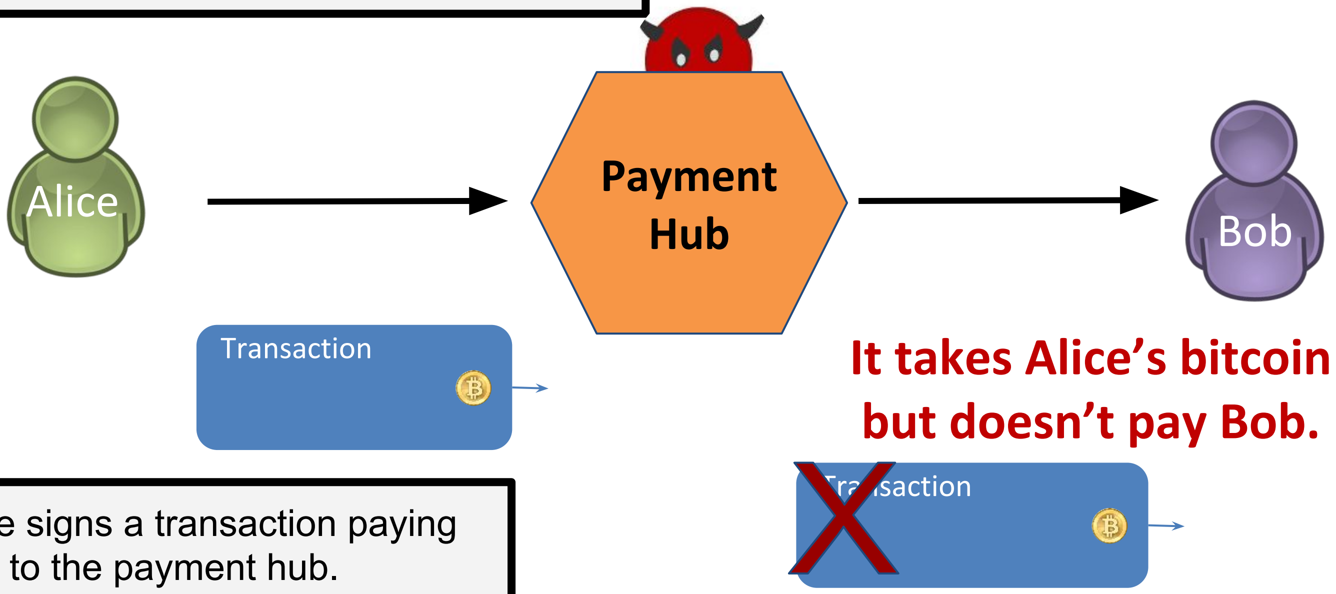


A Payment Hub solves these disadvantages.

Payment Hub: Details

But what if the payment hub is malicious and cheats Alice and Bob?

Alice wants to pays Bob via the payment hub.



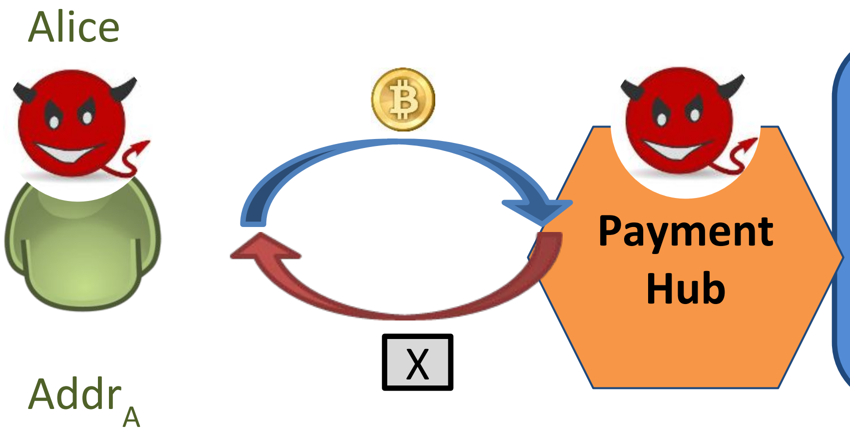
It takes Alice’s bitcoin but doesn’t pay Bob.

1. Alice signs a transaction paying 1 BTC to the payment hub.

2. Payment hub signs a transaction paying 1 BTC to Bob.

Bitcoin Transaction Contracts

Goal: **Fair Exchange/Atomic swaps:**



Alice

Addr$_A$

X

Payment Hub

Transaction Offer: X for ₿ .

“Alice pays ₿ to a spending transaction has a value X satisfying condition C.

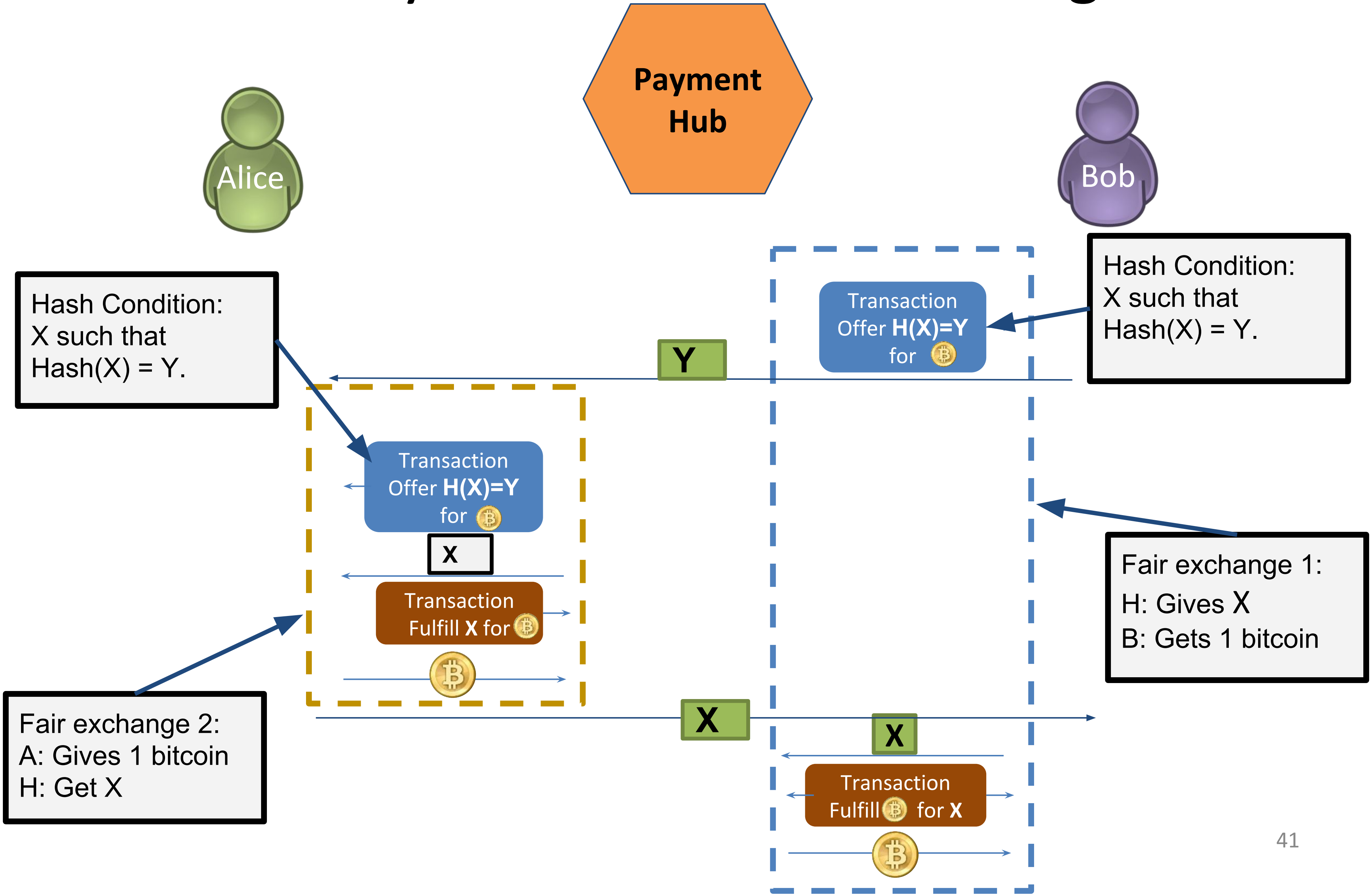
Transaction Fulfill: X for ₿ .

“Here is X .”

Bitcoin can only check two cryptographic conditions:

1. Hash(X) = Y,
2. Verify ECDSA Signature on a transaction.

Payment Hub: Fair Exchange



Payment Hub

Alice

Bob

Hash Condition:
X such that
Hash(X) = Y.

Transaction
Offer **H(X)=Y**
for ₿

Y

Hash Condition:
X such that
Hash(X) = Y.

Transaction
Offer **H(X)=Y**
for ₿

X

Transaction
Fulfill **X** for ₿

Fair exchange 1:
H: Gives X
B: Gets 1 bitcoin

Fair exchange 2:
A: Gives 1 bitcoin
H: Get X

X

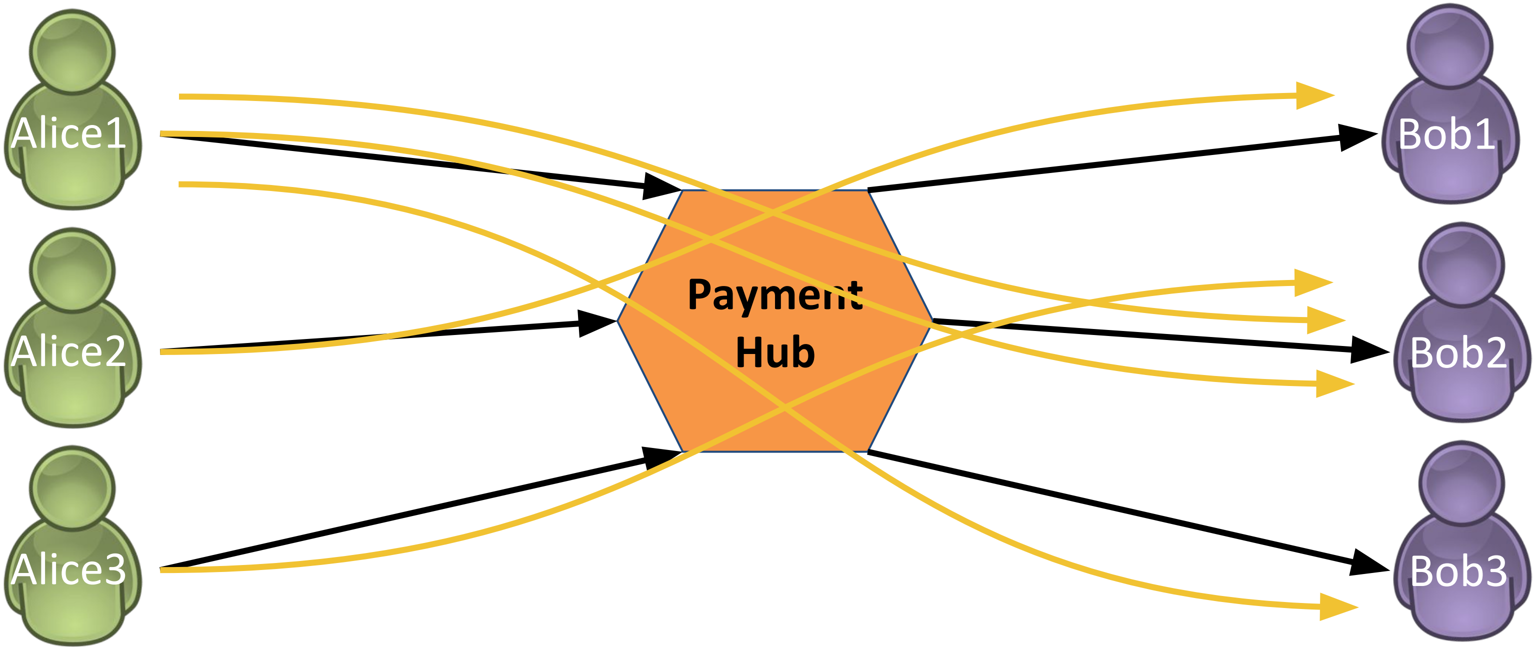
X

Transaction
Fulfill ₿ for **X**

Fair exchange prevents the Payment Hub from stealing.

Payment Hub: Privacy

While payment hubs are convenient, they do not offer any privacy against the payment hub.
The payment hub can trivially link the payer to the payee via the H(X)=Y used to ensure atomicity.



TumbleBit improves payment hubs so that
for each payment the payer can not be linked to the payees.

Outline

- Payment hubs
  - Bitcoin transactions/payment channels
  - What are Bitcoin payment hubs?
  - Scalability benefits of payment hubs
  - Are payment hubs private?
- TumbleBit as a Payment Hub
  - RSA-blind puzzles
  - TumbleBit as an unlinkable payment hub
  - Ensuring fair-exchange (TumbleBit can’t steal)
  - Puzzle-Promise-Protocol

RSA Puzzles

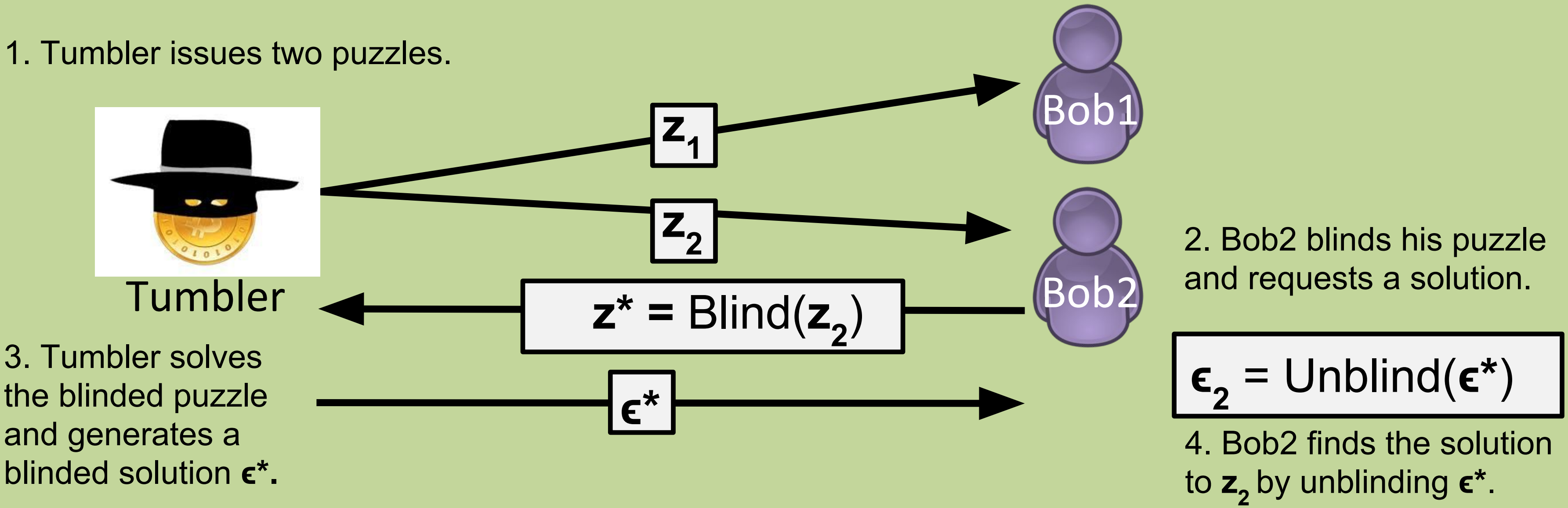
- An RSA Puzzle is just an RSA encryption of some value $\boldsymbol{\epsilon}$:

$$\mathbf{z} = \text{encRSA}(\boldsymbol{\epsilon}, \text{pk}) = \boldsymbol{\epsilon}^{\text{pk}} \text{ mod N}$$

- Only the party that knows sk can solve RSA puzzles:

$$\boldsymbol{\epsilon} = \text{decRSA}(\mathbf{z}, \text{sk}) = \mathbf{z}^{\text{sk}} \text{ mod N} = (\boldsymbol{\epsilon}^{\text{pk}})^{\text{sk}} \text{ mod N}$$

RSA blinding can be used to blind RSA puzzles



1. Tumbler issues two puzzles.

$\mathbf{z}^* = \text{Blind}(\mathbf{z}_2)$

2. Bob2 blinds his puzzle and requests a solution.

3. Tumbler solves the blinded puzzle and generates a blinded solution $\boldsymbol{\epsilon}^*$.

$\boldsymbol{\epsilon}_2 = \text{Unblind}(\boldsymbol{\epsilon}^*)$

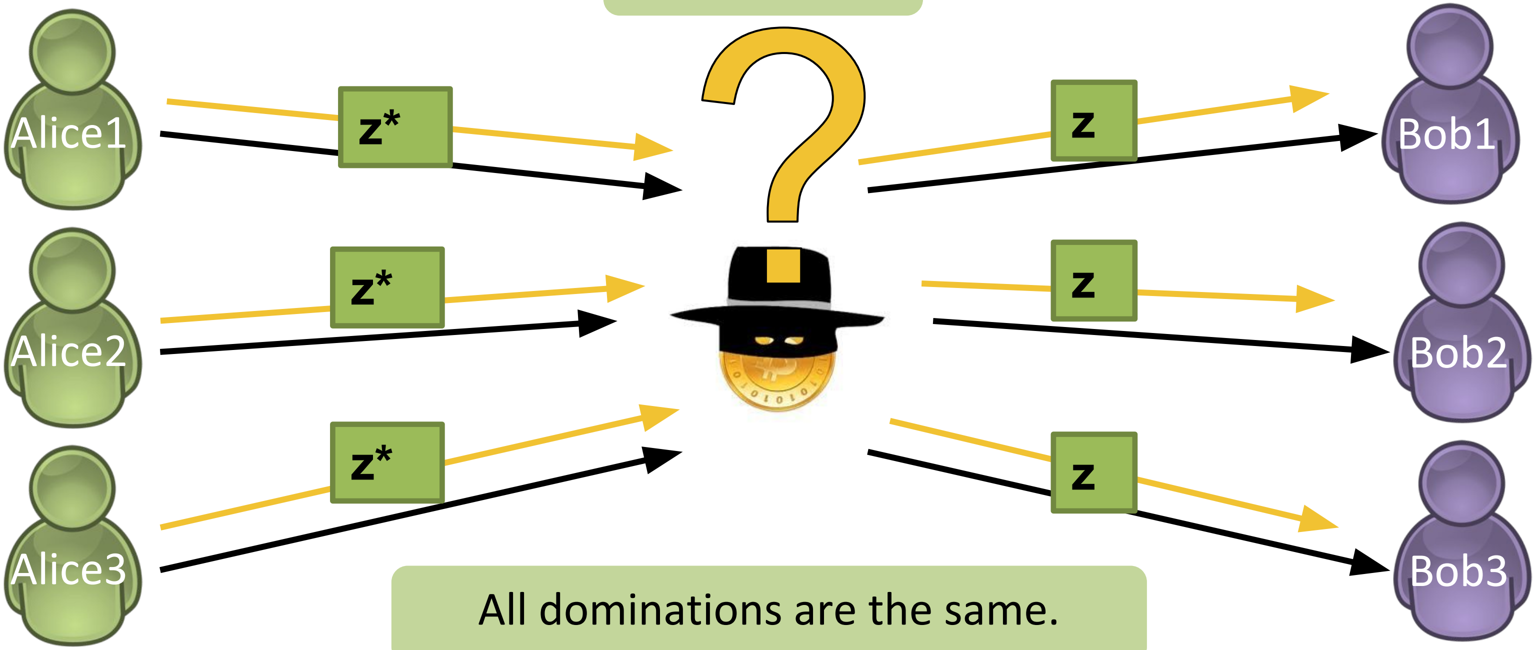
4. Bob2 finds the solution to $\mathbf{z}_2$ by unblinding $\boldsymbol{\epsilon}^*$.

Tumbler can not link the blinded RSA puzzle it solves to any of the RSA puzzles it issues.

Unlinkable Payments

We can use RSA puzzles to hide the link between payers and payees.

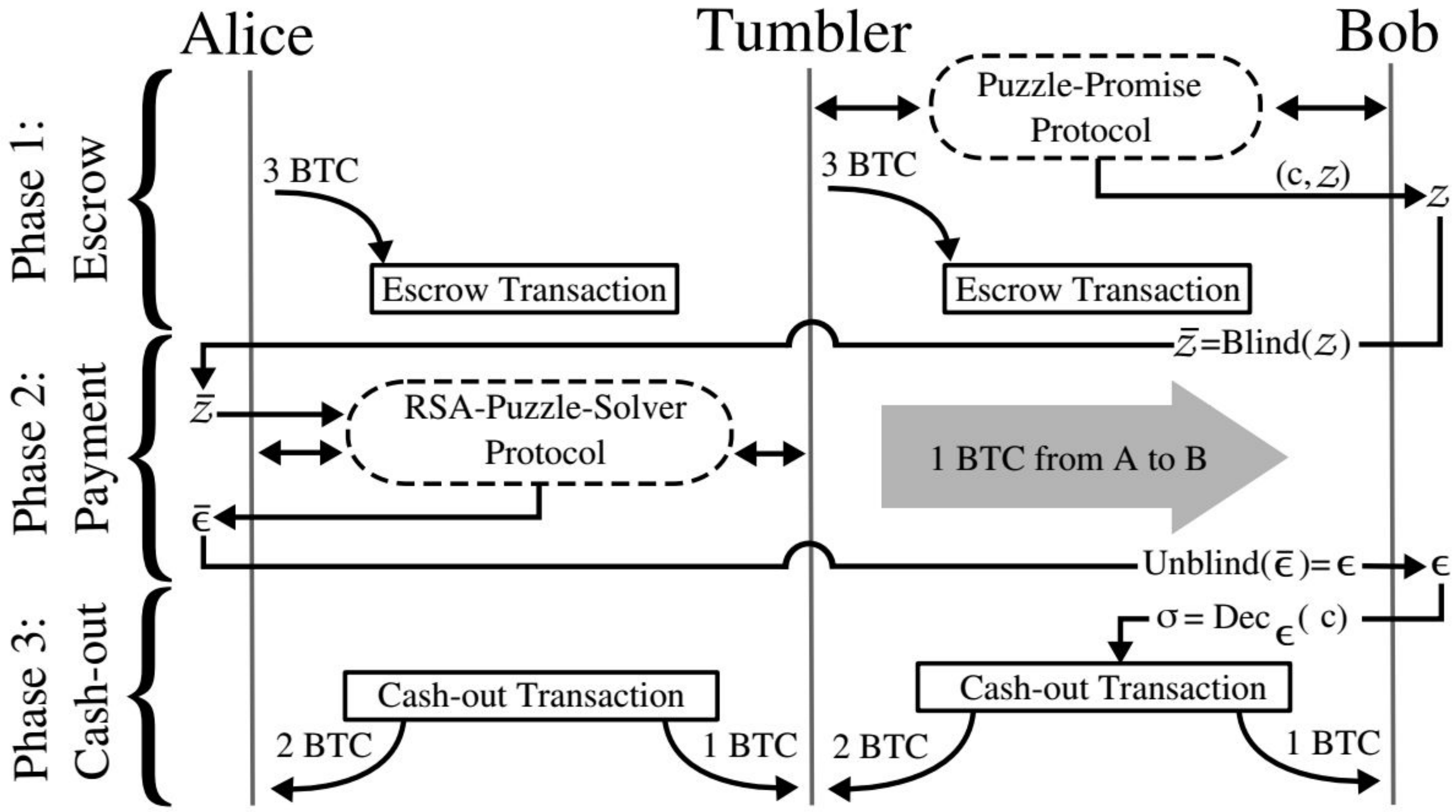
z* = Blind(**z**)



All dominations are the same.

...but how do we ensure that the tumbler does not cheat.

Puzzle-Promise-Protocol



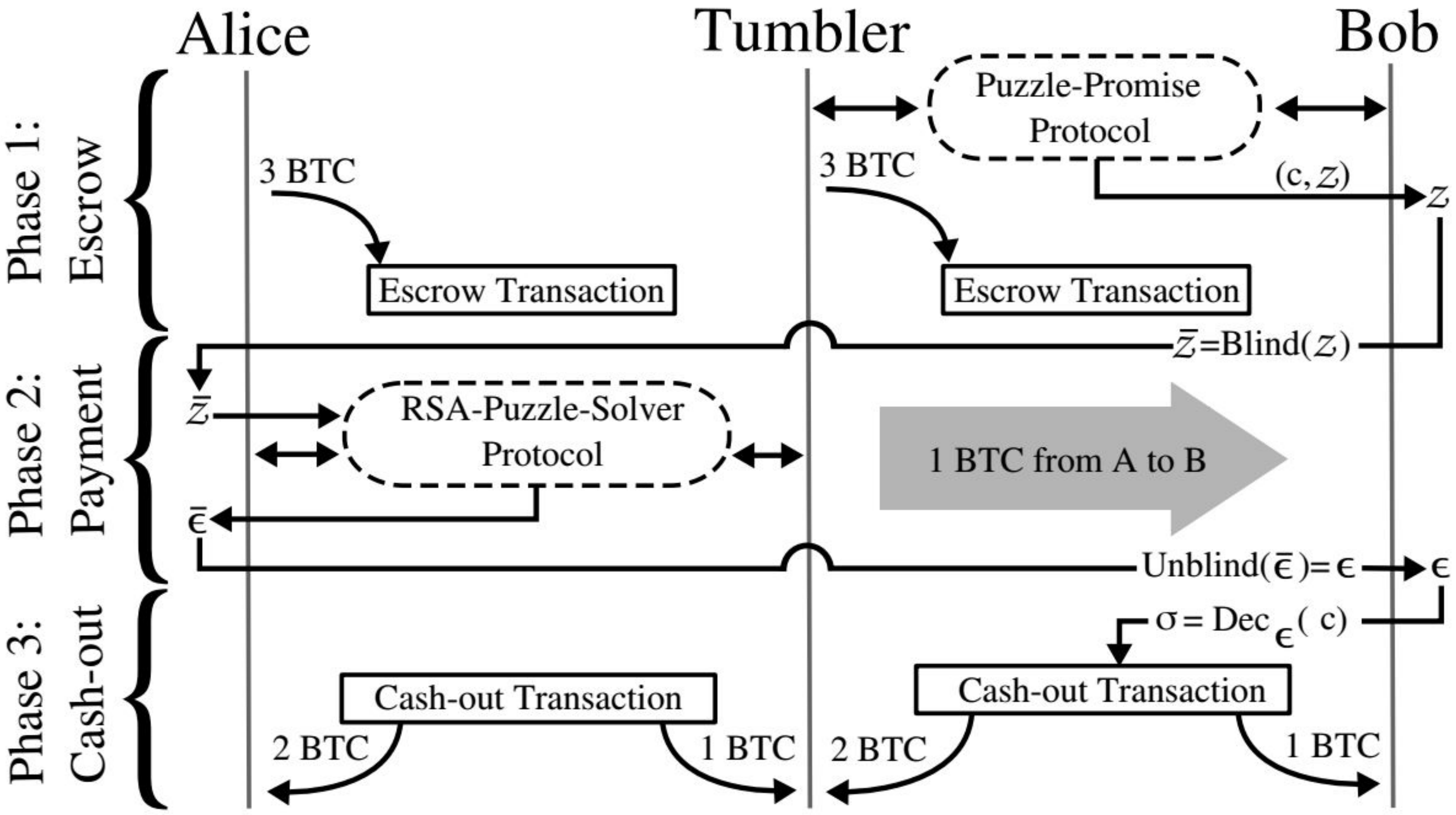
Many payments are made here.

Payments received are reflected here.

Payment unlinkability:

1. In payment: Tumbler can see that Alice paid (but no who she paid)
2. In cashout: Tumbler learns aggregate funds received by Bob.

Unlinkability



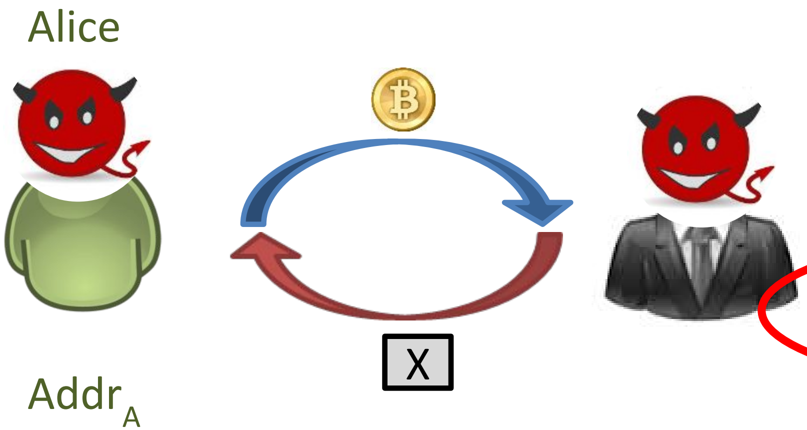
Many payments are made here.

Payments received are reflected here.

Payment unlinkability:
1. In payment: Tumbler can see that Alice paid (but no who she paid)
2. In cashout: Tumbler learns aggregate funds received by Bob.

Bitcoin Transaction Contracts

Goal: **Fair Exchange/Atomic swaps:**



Transaction Offer: X for ₿ .

“Addr_A pays ₿ to a spending transaction has a value X satisfying condition C.

Transaction Fulfill: X for ₿ .

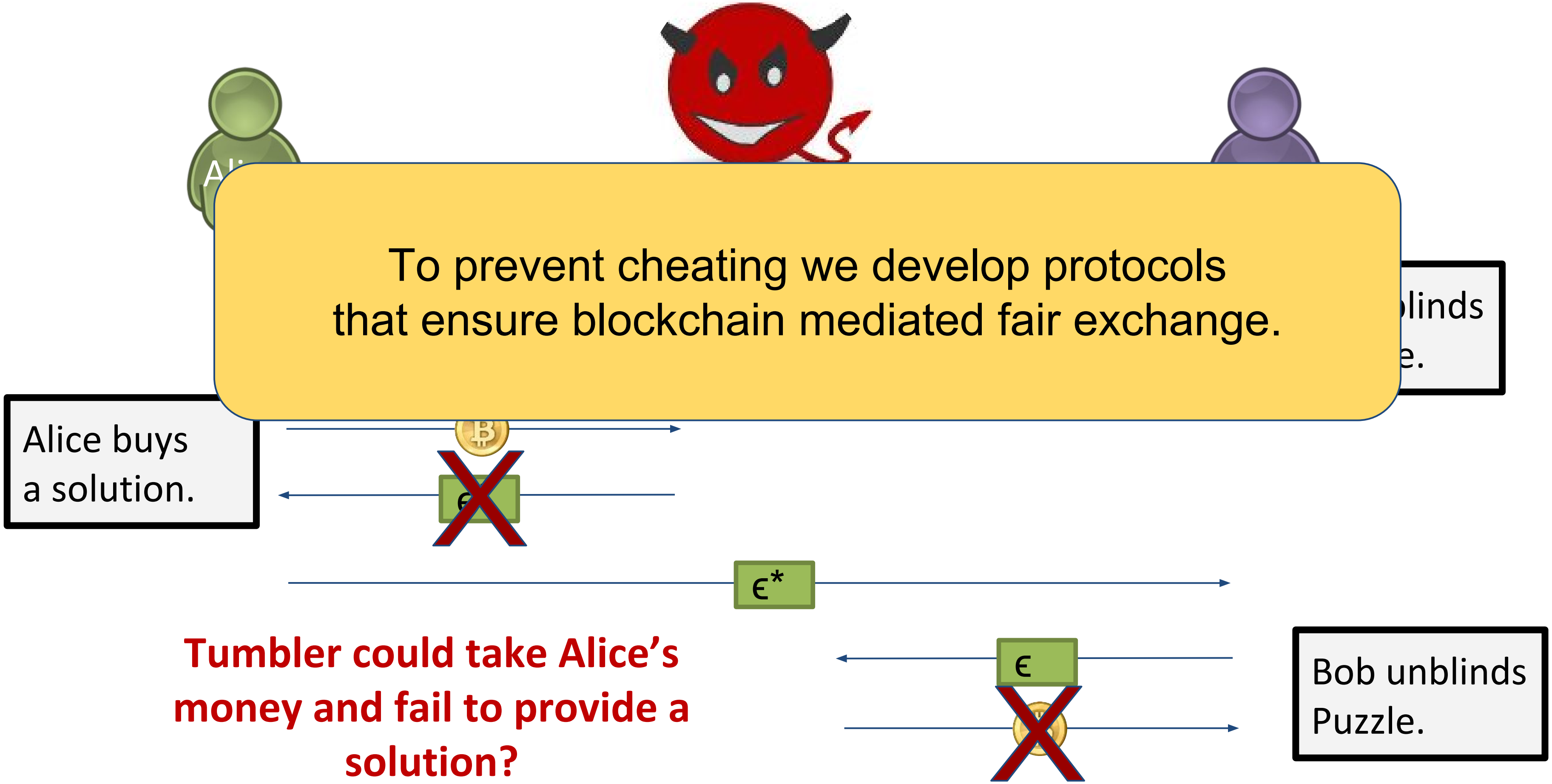
“Here is X .”

Bitcoin transaction scripts are very limited.
We can only check two types of cryptographic conditions C:

1. Hash(X) = Y,
2. ECDSA_CheckSignature(Tx, PUBLIC_KEY) = TRUE

TumbleBit: Paying with RSA-Puzzles

But what if the Tumbler is malicious and cheats Alice and Bob?



To prevent cheating we develop protocols that ensure blockchain mediated fair exchange.

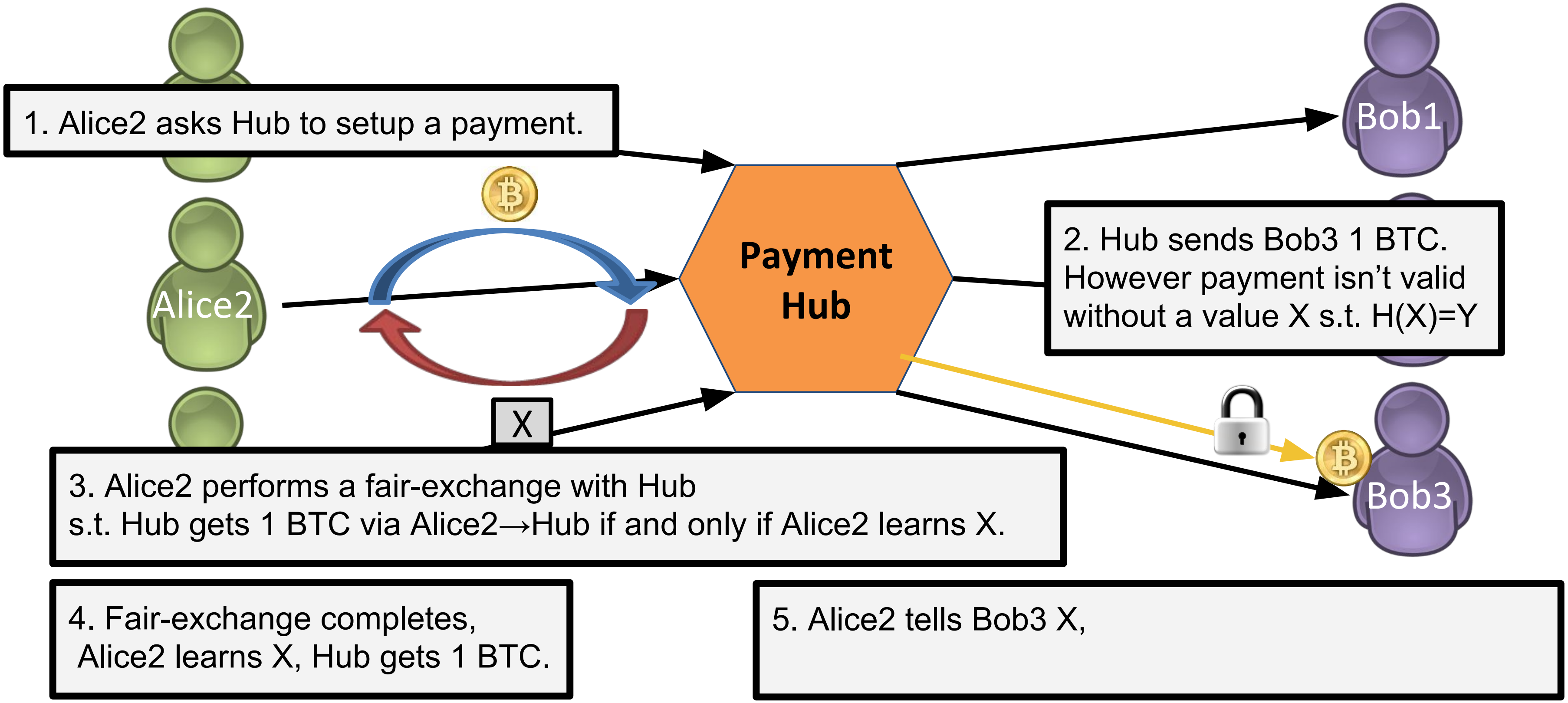
Alice buys a solution.

Tumbler could take Alice’s money and fail to provide a solution?

Bob unblinds Puzzle.

Tumbler could refuse to pay for a solution?

Payment Hubs: Preventing Theft



1. Alice2 asks Hub to setup a payment.

2. Hub sends Bob3 1 BTC.
However payment isn’t valid
without a value X s.t. H(X)=Y

3. Alice2 performs a fair-exchange with Hub
s.t. Hub gets 1 BTC via Alice2→Hub if and only if Alice2 learns X.

4. Fair-exchange completes,
Alice2 learns X, Hub gets 1 BTC.

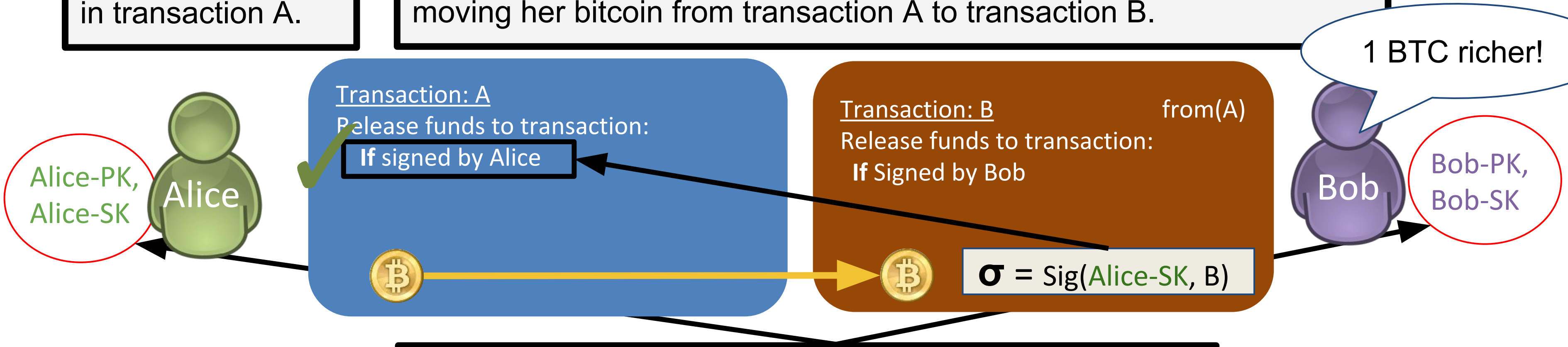
5. Alice2 tells Bob3 X,

We want to ensure that the transaction Alice2→Hub is atomic with Hub→Bob3.

Background: Bitcoin Transactions

1. Alice has 1 BTC in transaction A.

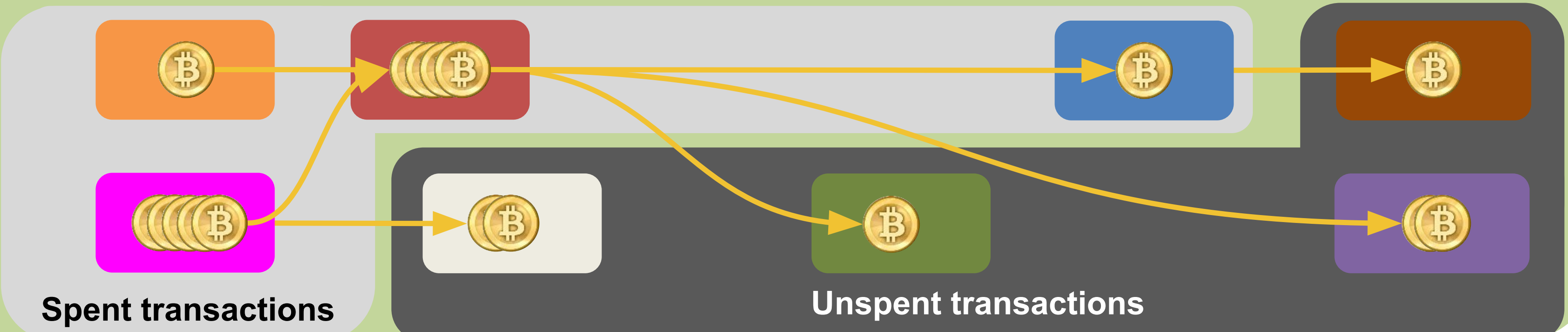
2. To setup a payment to Bob of 1 BTC Alice creates a transaction B moving her bitcoin from transaction A to transaction B.



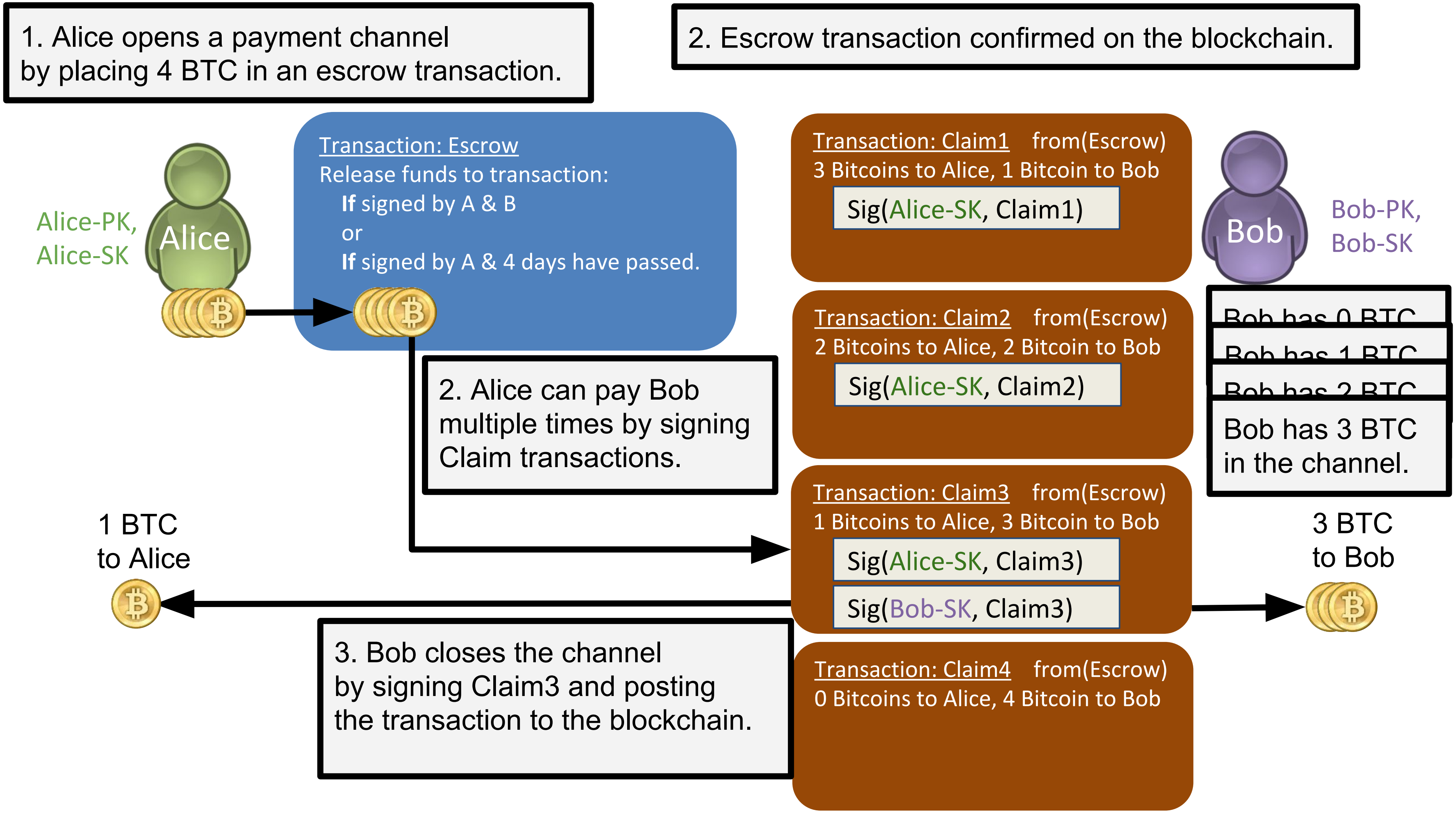
3. Alice then signs transaction B to fulfill A's condition...

...thus transferring that bitcoin from transaction A to transaction B.

Payment in Bitcoin occurs by transferring bitcoins in one transaction to a new transaction...
...thus, ownership is merely holding a secret key which can authorize such transfers.



Background: Payment Hub



1. Alice opens a payment channel by placing 4 BTC in an escrow transaction.

2. Escrow transaction confirmed on the blockchain.

Transaction: Escrow
Release funds to transaction:
If signed by A & B
or
If signed by A & 4 days have passed.

2. Alice can pay Bob multiple times by signing Claim transactions.

Transaction: Claim1 from(Escrow)
3 Bitcoins to Alice, 1 Bitcoin to Bob
Sig(Alice-SK, Claim1)

Transaction: Claim2 from(Escrow)
2 Bitcoins to Alice, 2 Bitcoin to Bob
Sig(Alice-SK, Claim2)

Transaction: Claim3 from(Escrow)
1 Bitcoins to Alice, 3 Bitcoin to Bob
Sig(Alice-SK, Claim3)
Sig(Bob-SK, Claim3)

Transaction: Claim4 from(Escrow)
0 Bitcoins to Alice, 4 Bitcoin to Bob

Bob has 0 BTC

Bob has 1 BTC

Bob has 2 BTC

Bob has 3 BTC in the channel.

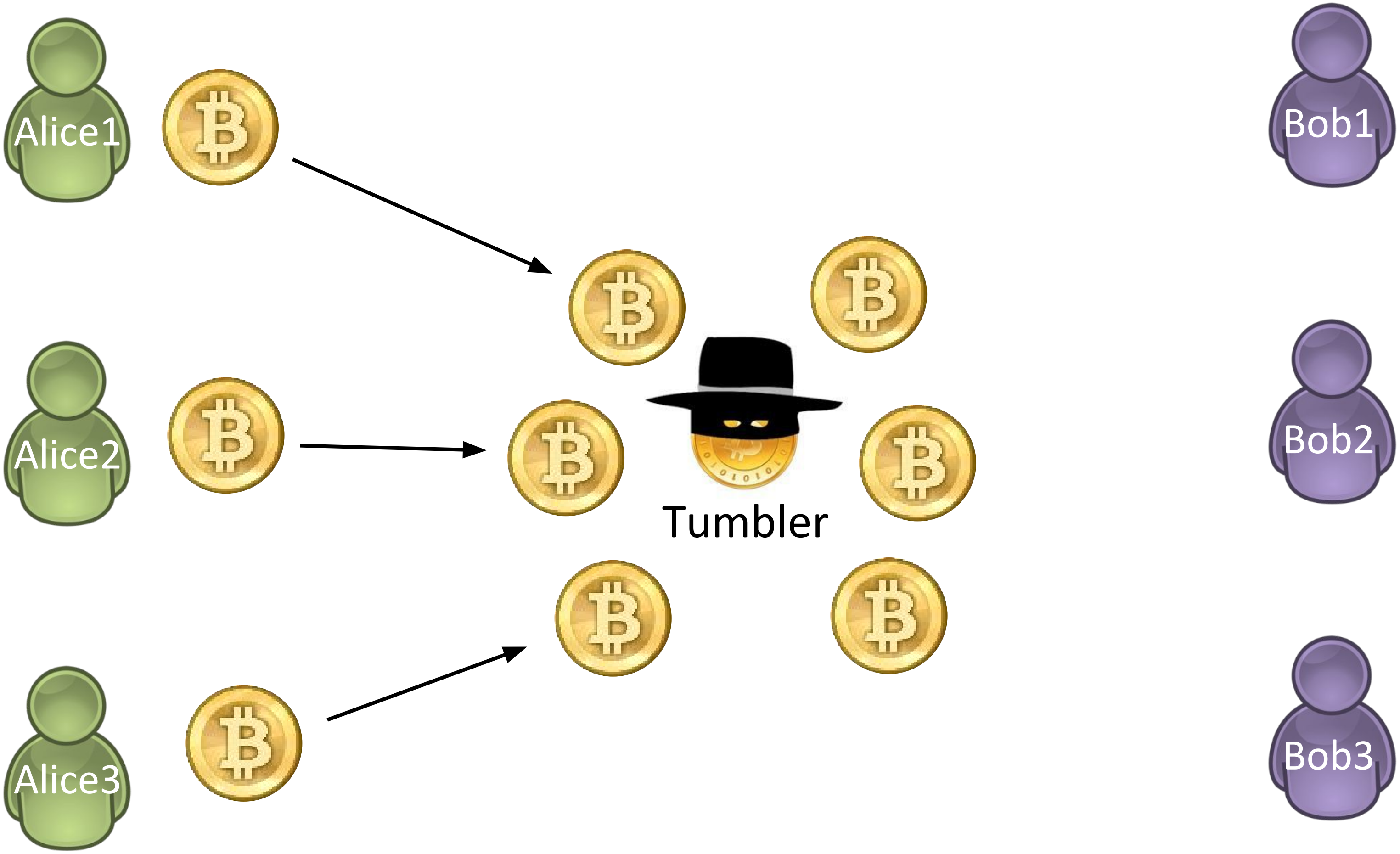
1 BTC to Alice

3 BTC to Bob

3. Bob closes the channel by signing Claim3 and posting the transaction to the blockchain.

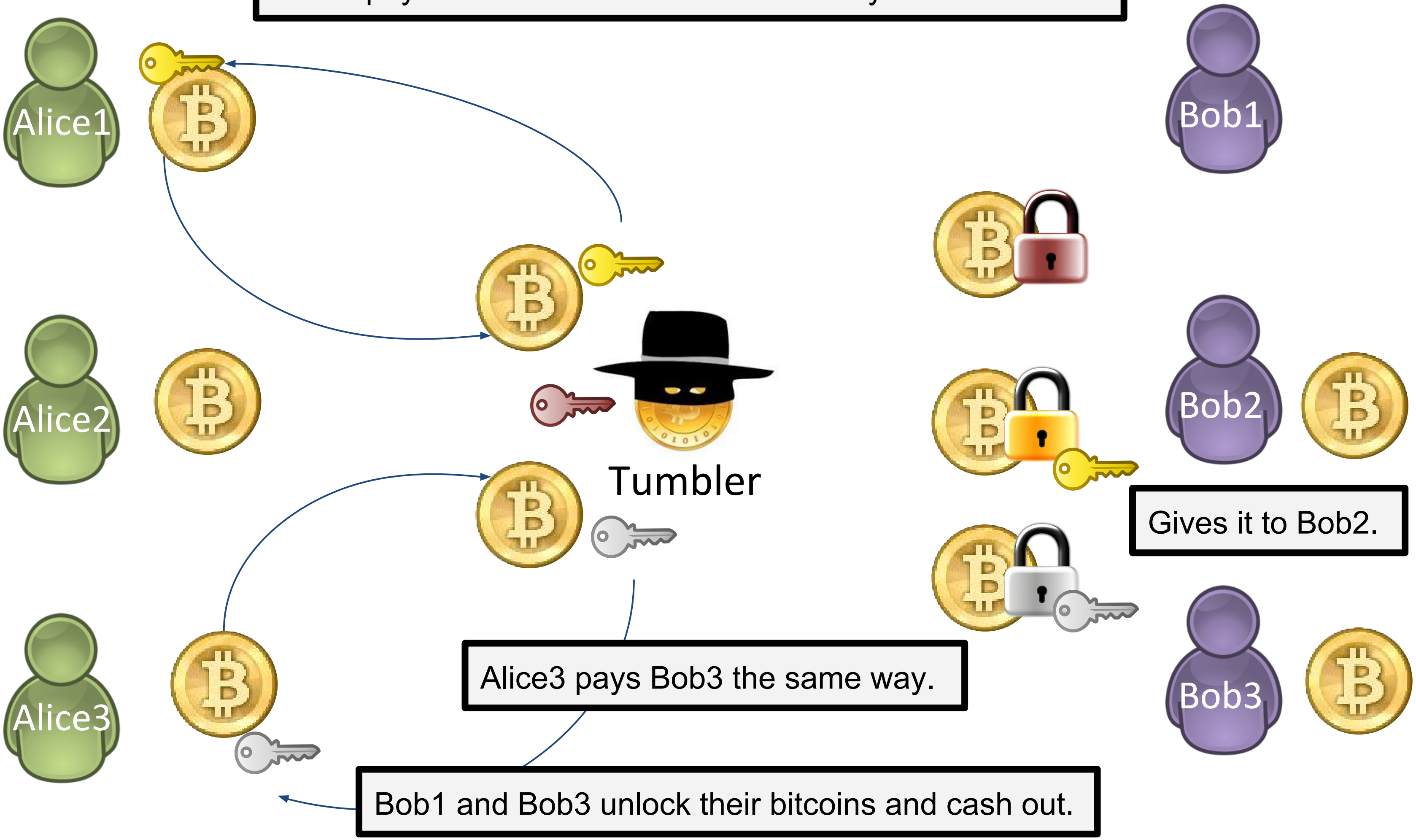
Alice was able to make N instant transactions to Bob.

TumbleBit: The Basic Idea



Intuition: Tumbler gives out locked bitcoins and sells keys.

TumbleBit: Overview



Alice1 pays the tumbler 1 Bitcoin for the key to Bob2’s lock.

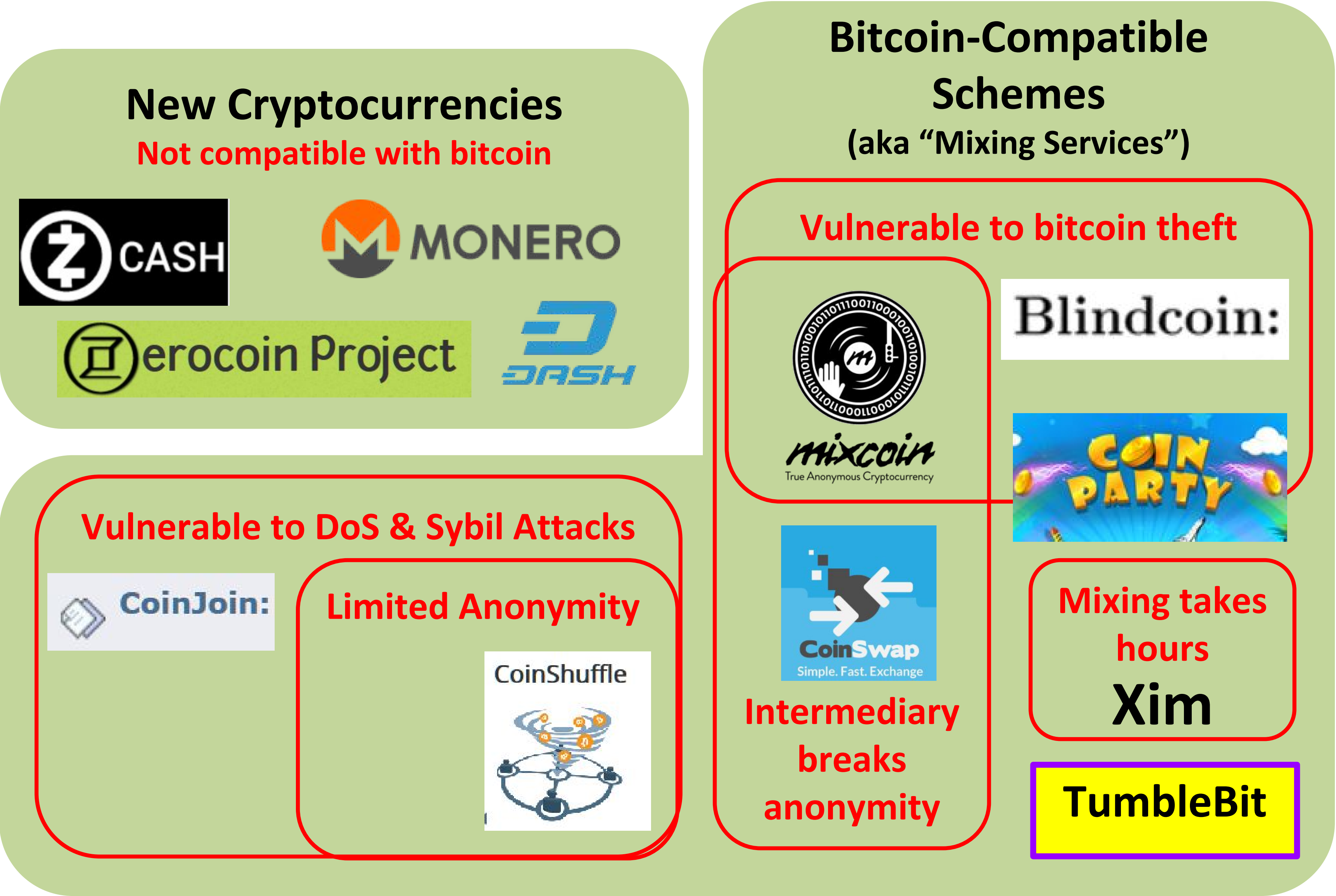
Gives it to Bob2.

Alice3 pays Bob3 the same way.

Bob1 and Bob3 unlock their bitcoins and cash out.

Intuition: Tumbler gives out locked bitcoins and sells keys.

Related Work



New Cryptocurrencies

Not compatible with bitcoin

CASH

MONERO

erocoin Project

DASH

Bitcoin-Compatible Schemes

(aka “Mixing Services”)

Vulnerable to bitcoin theft

mixcoin

True Anonymous Cryptocurrency

Blindcoin:

COIN PARTY

CoinSwap

Simple. Fast. Exchange

Intermediary breaks anonymity

Mixing takes hours

Xim

TumbleBit

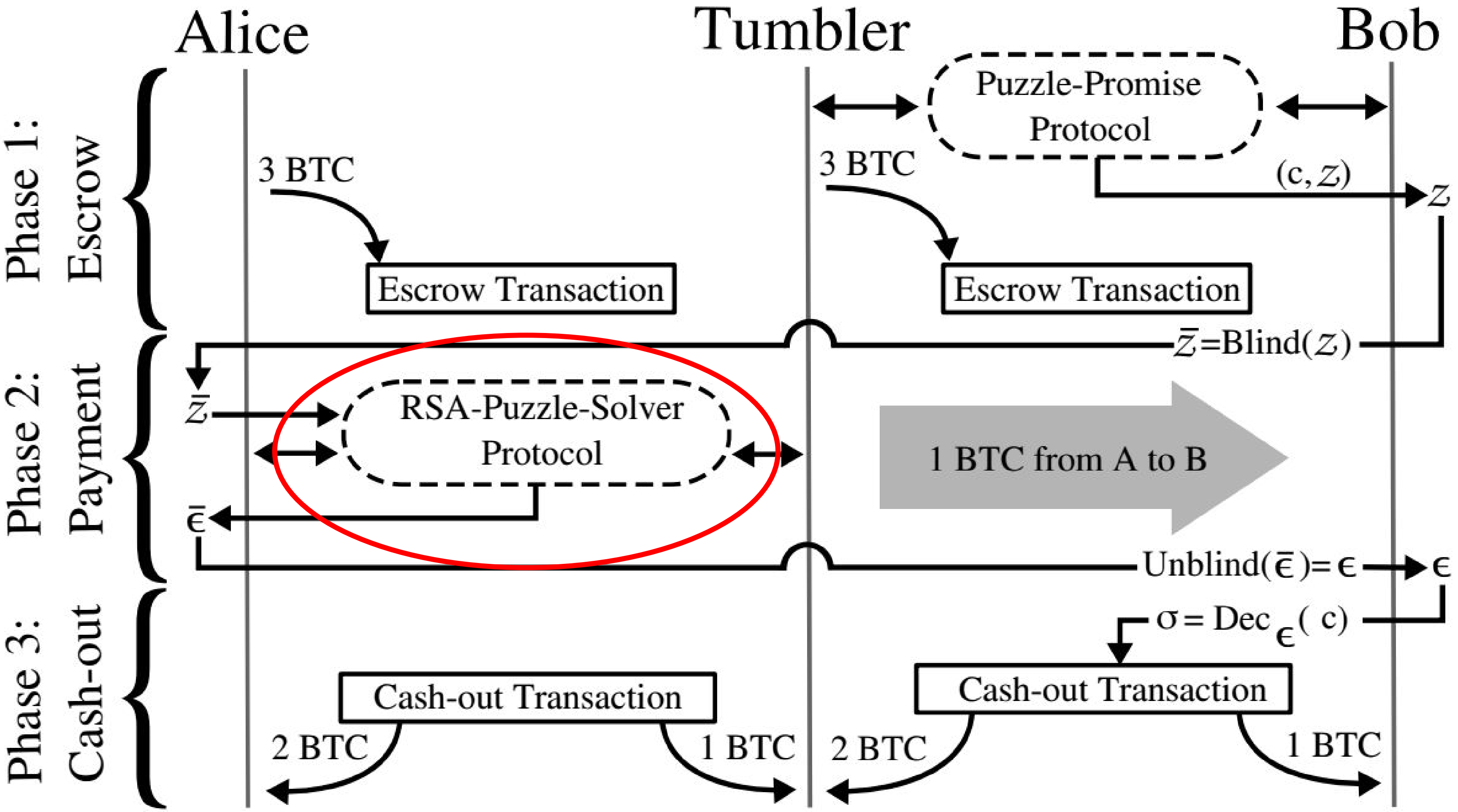
Vulnerable to DoS & Sybil Attacks

CoinJoin:

Limited Anonymity

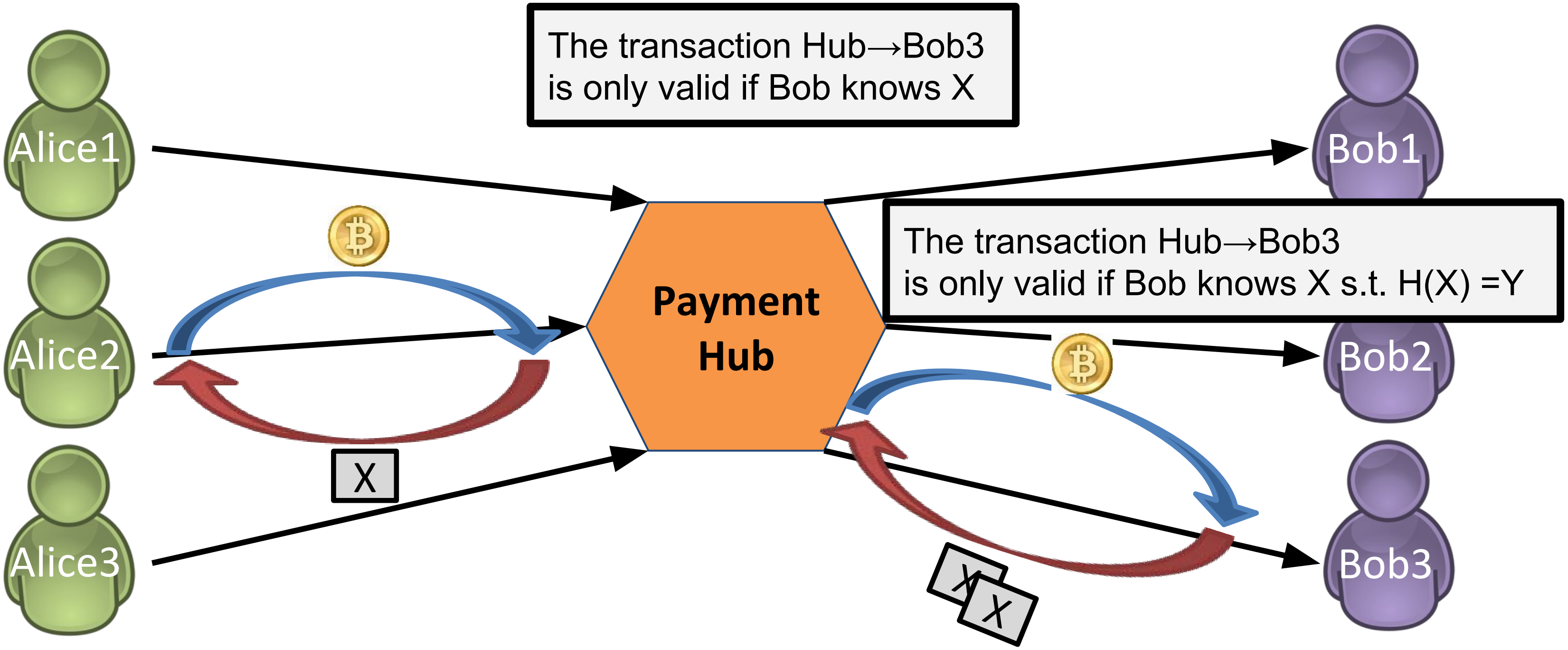
CoinShuffle

RSA-Puzzle-Solver Protocol



I'm only going to walk through the RSA-Puzzle-Solver Protocol, but it is similar to the Puzzle-Promise-Protocol.

Payment Hubs: Preventing Theft



The transaction Hub→Bob3
is only valid if Bob knows X

The transaction Hub→Bob3
is only valid if Bob knows X s.t. H(X) =Y

We want to ensure that the transaction Alice2→Hub is atomic with Hub→Bob3.

Puzzle-Solver-Protocol

Blinded puzzle

$\boxed{z^*} = y$

Alice mixes the puzzle
she want solved with
fake puzzles for y
she know the so

Alice reveals the
puzzles and ask
Tumbler to oper
commitments.

Alice checks tha
fake puzzle
commitments op
correct values.

Alice learns the
solution to y and
sends it to Bob.

The Tumbler uses this protocol to convince Alice

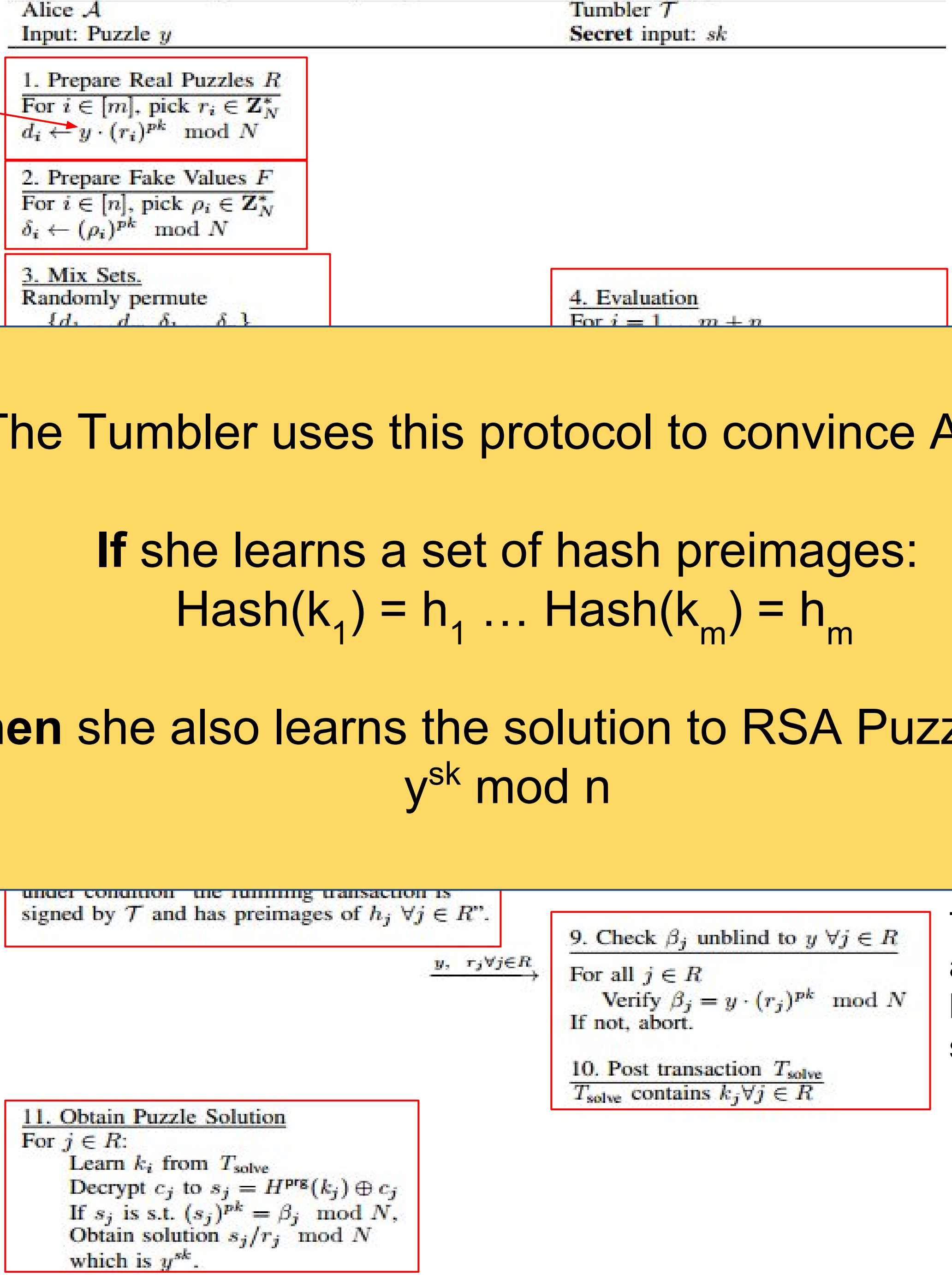
If she learns a set of hash preimages:
$\mathrm{Hash}(k_1) = h_1 \ldots \mathrm{Hash}(k_m) = h_m$

Then she also learns the solution to RSA Puzzle y:
$y^{sk} \bmod n$

Tumbler solves all the
crypts the
l commits

cks that
all the
he fake
then
itments.

Tumbler ensures that
all the real puzzles
have the same
solution.



Alice $\mathcal{A}$
Input: Puzzle y

Tumbler $\mathcal{T}$
Secret input: sk

1. Prepare Real Puzzles R
For $i \in [m]$, pick $r_i \in \mathbf{Z}_N^*$
$d_i \leftarrow y \cdot (r_i)^{pk} \mod N$

2. Prepare Fake Values F
For $i \in [n]$, pick $\rho_i \in \mathbf{Z}_N^*$
$\delta_i \leftarrow (\rho_i)^{pk} \mod N$

3. Mix Sets.
Randomly permute

4. Evaluation
For $i = 1, \ldots, m+n$

signed by $\mathcal{T}$ and has preimages of $h_j \; \forall j \in R$".

$\xrightarrow{y,\ r_j \forall j \in R}$

9. Check β_j unblind to $y \; \forall j \in R$
For all $j \in R$
Verify $\beta_j = y \cdot (r_j)^{pk} \mod N$
If not, abort.

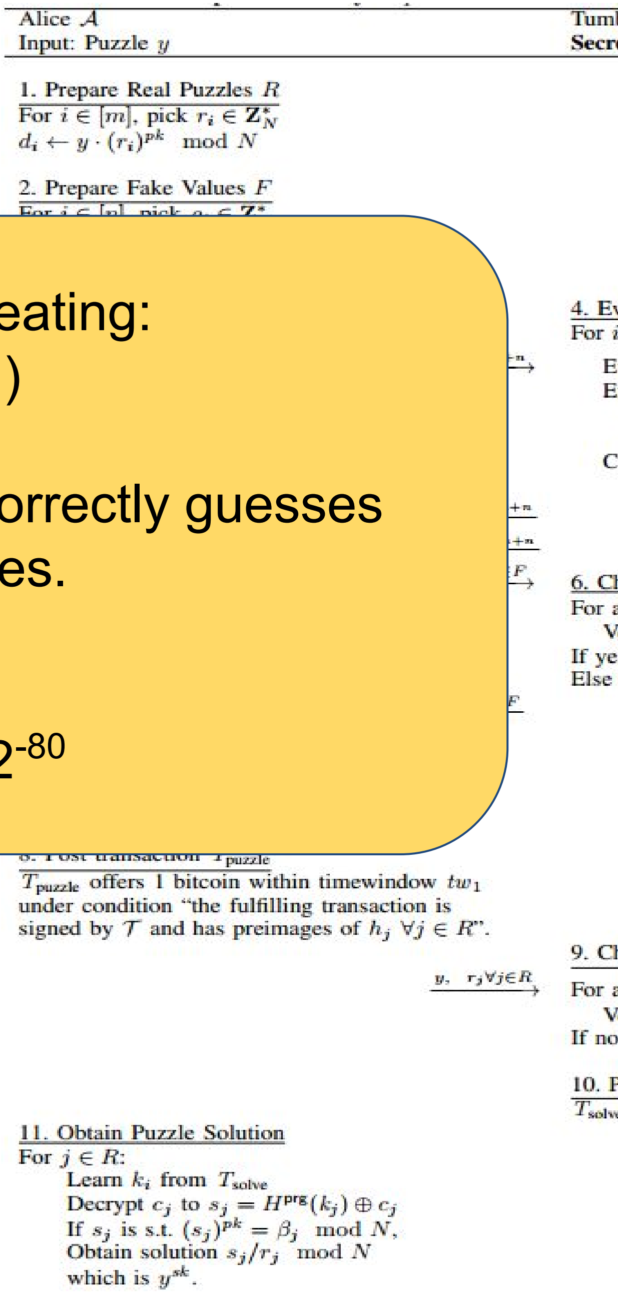
10. Post transaction T_{solve}
T_{solve} contains $k_j \forall j \in R$

11. Obtain Puzzle Solution
For $j \in R$:
Learn k_i from T_{solve}
Decrypt c_j to $s_j = H^{\mathsf{prg}}(k_j) \oplus c_j$
If s_j is s.t. $(s_j)^{pk} = \beta_j \mod N$,
Obtain solution $s_j / r_j \mod N$
which is y^{sk}.

Security of Puzzle-Solver-Protocol

M = size of real set →

N = size of fake set →



Alice $\mathcal{A}$
Input: Puzzle y

1. Prepare Real Puzzles R
For $i \in [m]$, pick $r_i \in \mathbf{Z}_N^*$
$d_i \leftarrow y \cdot (r_i)^{pk} \mod N$

2. Prepare Fake Values F

T_{puzzle} offers 1 bitcoin within timewindow tw_1 under condition "the fulfilling transaction is signed by $\mathcal{T}$ and has preimages of h_j $\forall j \in R$".

$\xrightarrow{y,\ r_j \forall j \in R}$

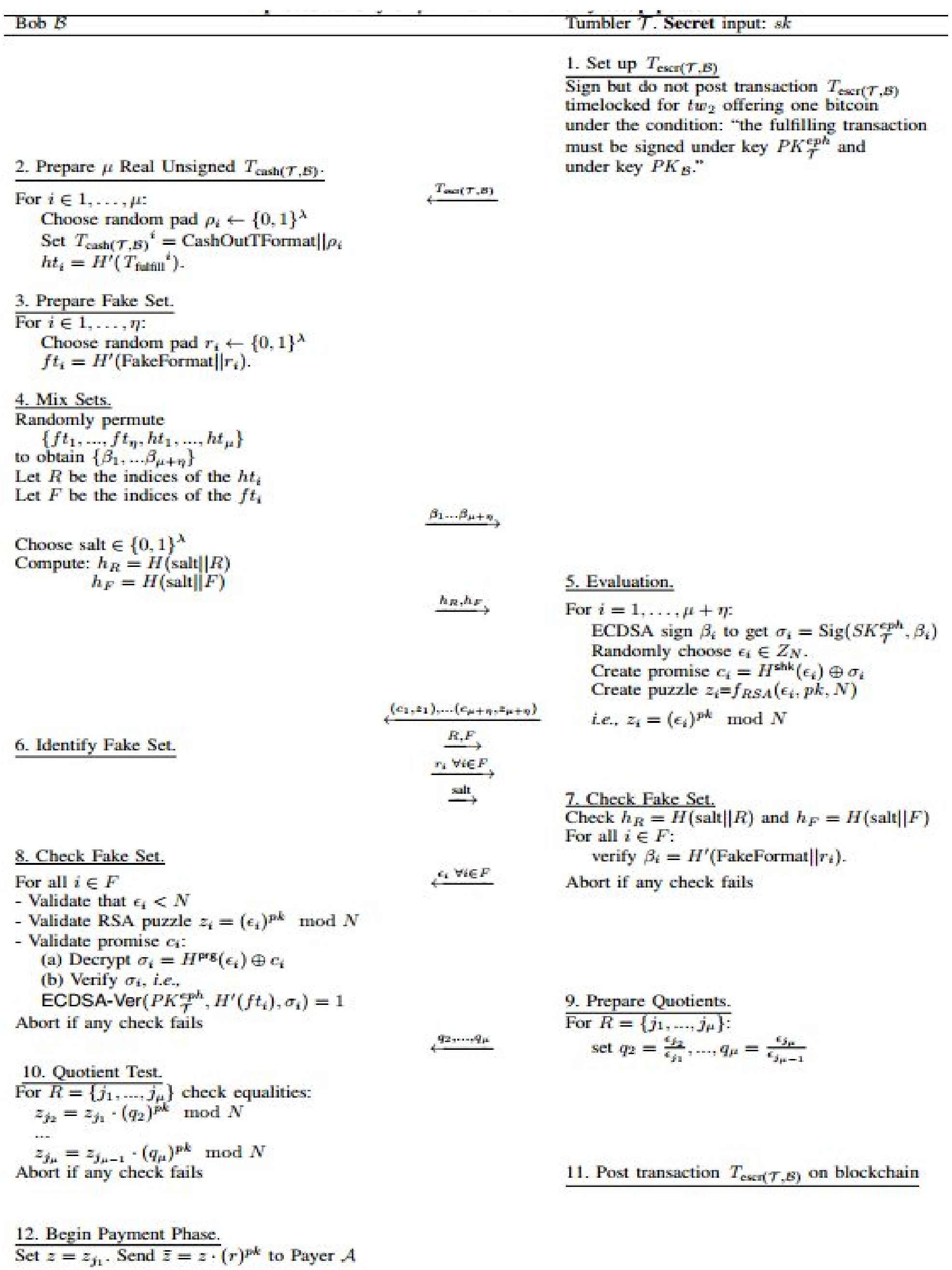
11. Obtain Puzzle Solution
For $j \in R$:
Learn k_i from T_{solve}
Decrypt c_j to $s_j = H^{\text{prg}}(k_j) \oplus c_j$
If s_j is s.t. $(s_j)^{pk} = \beta_j \mod N$,
Obtain solution $s_j / r_j \mod N$
which is y^{sk}.

Prob of the Tumbler cheating:
1/(M+N choose M)
or
the probability that the Tumbler correctly guesses the real set of puzzles.

M = 15, N = 285
Prob of cheating = 2^{-80}

- The Tumbl
  Alice is c
  AND
  $k_1 \ldots k_m$ **c**
- Two paran
  M and N
- If the Tumb
  … Alice wi
- If the Tumbler corrupts >M solutions:
  … Alice will always detect cheating.
- The Tumbler must corrupt exactly M solutions
  … and must only corrupt the real set.

Puzzle-Promise-Protocol



Bob $\mathcal{B}$

Tumbler $\mathcal{T}$. **Secret input:** sk

1. Set up $T_{\text{escr}(\mathcal{T},\mathcal{B})}$

Sign but do not post transaction $T_{\text{escr}(\mathcal{T},\mathcal{B})}$ timelocked for tw_2 offering one bitcoin under the condition: "the fulfilling transaction must be signed under key $PK^{eph}_{\mathcal{T}}$ and under key $PK_{\mathcal{B}}$."

$\xleftarrow{T_{\text{escr}(\mathcal{T},\mathcal{B})}}$

2. Prepare μ Real Unsigned $T_{\text{cash}(\mathcal{T},\mathcal{B})}$.

For $i \in 1, \ldots, \mu$:
  Choose random pad $\rho_i \leftarrow \{0,1\}^\lambda$
  Set $T_{\text{cash}(\mathcal{T},\mathcal{B})}{}^i = \text{CashOutTFormat}||\rho_i$
  $ht_i = H'(T_{\text{fulfill}}{}^i)$.

3. Prepare Fake Set.

For $i \in 1, \ldots, \eta$:
  Choose random pad $r_i \leftarrow \{0,1\}^\lambda$
  $ft_i = H'(\text{FakeFormat}||r_i)$.

4. Mix Sets.

Randomly permute
  $\{ft_1, ..., ft_\eta, ht_1, ..., ht_\mu\}$
to obtain $\{\beta_1, ...\beta_{\mu+\eta}\}$
Let R be the indices of the ht_i
Let F be the indices of the ft_i

$\xrightarrow{\beta_1 \ldots \beta_{\mu+\eta}}$

Choose salt $\in \{0,1\}^\lambda$
Compute: $h_R = H(\text{salt}||R)$
  $h_F = H(\text{salt}||F)$

$\xrightarrow{h_R, h_F}$

5. Evaluation.

For $i = 1, \ldots, \mu + \eta$:
  ECDSA sign β_i to get $\sigma_i = \text{Sig}(SK^{eph}_{\mathcal{T}}, \beta_i)$
  Randomly choose $\epsilon_i \in Z_N$.
  Create promise $c_i = H^{\text{shk}}(\epsilon_i) \oplus \sigma_i$
  Create puzzle $z_i = f_{RSA}(\epsilon_i, pk, N)$
  i.e., $z_i = (\epsilon_i)^{pk} \mod N$

$\xleftarrow{(c_1, z_1), \ldots (c_{\mu+\eta}, z_{\mu+\eta})}$

6. Identify Fake Set.

$\xrightarrow{R, F}$

$\xrightarrow{r_i \ \forall i \in F}$

$\xrightarrow{\text{salt}}$

7. Check Fake Set.

Check $h_R = H(\text{salt}||R)$ and $h_F = H(\text{salt}||F)$
For all $i \in F$:
  verify $\beta_i = H'(\text{FakeFormat}||r_i)$.
Abort if any check fails

$\xleftarrow{\epsilon_i \ \forall i \in F}$

8. Check Fake Set.

For all $i \in F$
- Validate that $\epsilon_i < N$
- Validate RSA puzzle $z_i = (\epsilon_i)^{pk} \mod N$
- Validate promise c_i:
  (a) Decrypt $\sigma_i = H^{\text{prg}}(\epsilon_i) \oplus c_i$
  (b) Verify σ_i, *i.e.*,
  $\text{ECDSA-Ver}(PK^{eph}_{\mathcal{T}}, H'(ft_i), \sigma_i) = 1$

Abort if any check fails

9. Prepare Quotients.

For $R = \{j_1, ..., j_\mu\}$:
  set $q_2 = \frac{\epsilon_{j_2}}{\epsilon_{j_1}}, ..., q_\mu = \frac{\epsilon_{j_\mu}}{\epsilon_{j_{\mu-1}}}$

$\xleftarrow{q_2, \ldots, q_\mu}$

10. Quotient Test.

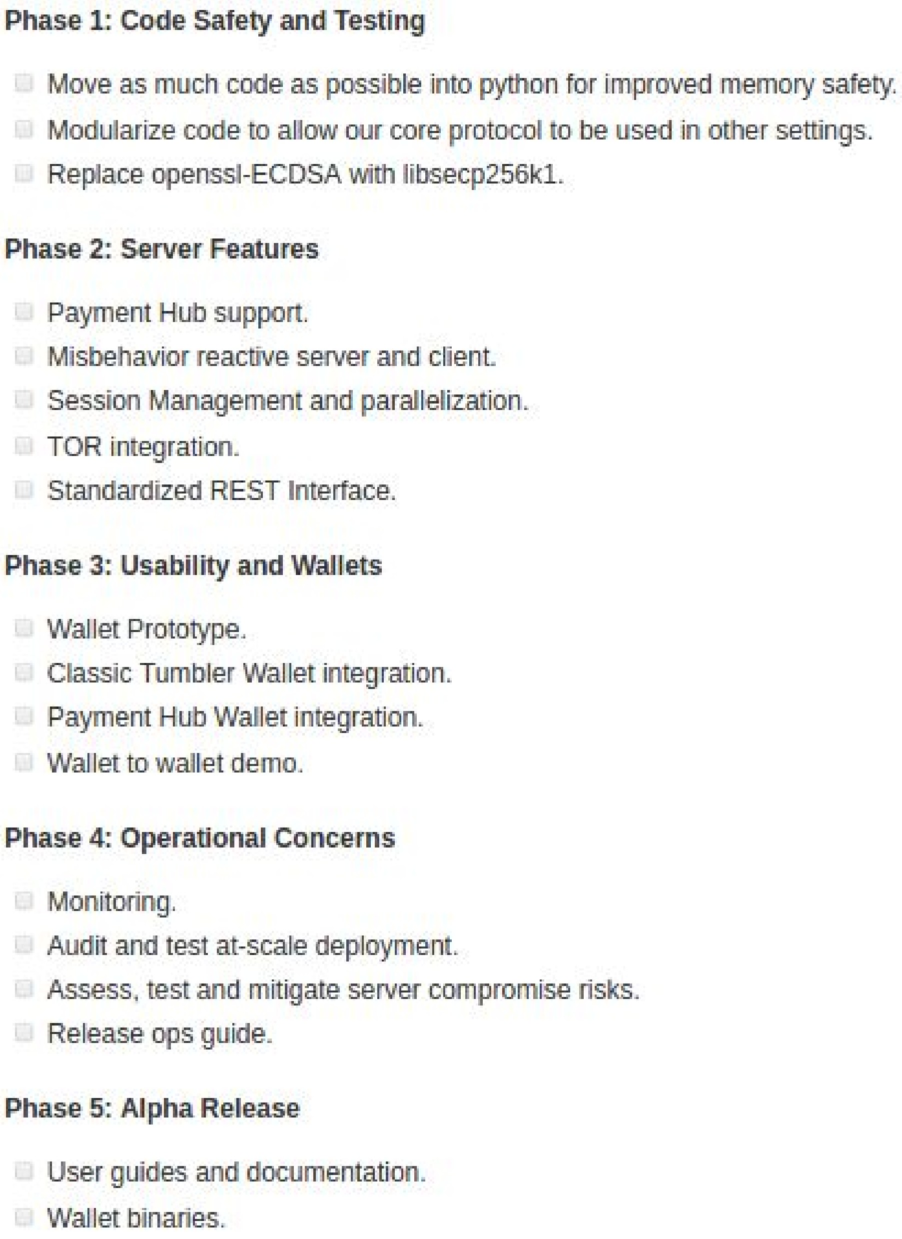
For $R = \{j_1, ..., j_\mu\}$ check equalities:
  $z_{j_2} = z_{j_1} \cdot (q_2)^{pk} \mod N$
  ...
  $z_{j_\mu} = z_{j_{\mu-1}} \cdot (q_\mu)^{pk} \mod N$
Abort if any check fails

11. Post transaction $T_{\text{escr}(\mathcal{T},\mathcal{B})}$ on blockchain

12. Begin Payment Phase.

Set $z = z_{j_1}$. Send $\overline{z} = z \cdot (r)^{pk}$ to Payer $\mathcal{A}$

TumbleBit: Roadmap



Phase 1: Code Safety and Testing

- ☐ Move as much code as possible into python for improved memory safety.
- ☐ Modularize code to allow our core protocol to be used in other settings.
- ☐ Replace openssl-ECDSA with libsecp256k1.

Phase 2: Server Features

- ☐ Payment Hub support.
- ☐ Misbehavior reactive server and client.
- ☐ Session Management and parallelization.
- ☐ TOR integration.
- ☐ Standardized REST Interface.

Phase 3: Usability and Wallets

- ☐ Wallet Prototype.
- ☐ Classic Tumbler Wallet integration.
- ☐ Payment Hub Wallet integration.
- ☐ Wallet to wallet demo.

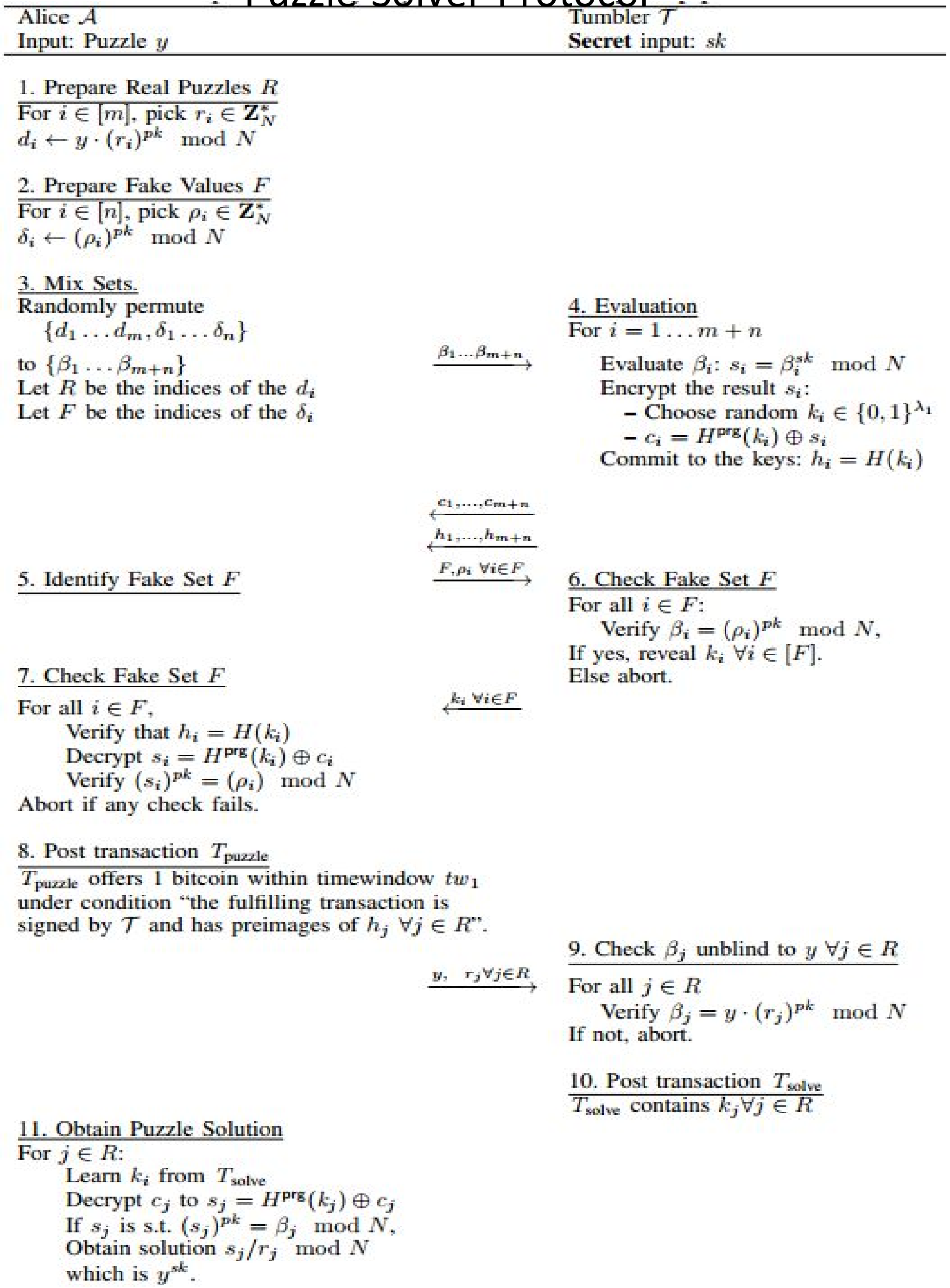
Phase 4: Operational Concerns

- ☐ Monitoring.
- ☐ Audit and test at-scale deployment.
- ☐ Assess, test and mitigate server compromise risks.
- ☐ Release ops guide.

Phase 5: Alpha Release

- ☐ User guides and documentation.
- ☐ Wallet binaries.

Puzzle-Solver-Protocol



Alice $\mathcal{A}$
Input: Puzzle y

Tumbler $\mathcal{T}$
Secret input: sk

1. Prepare Real Puzzles R
For $i \in [m]$, pick $r_i \in \mathbf{Z}_N^*$
$d_i \leftarrow y \cdot (r_i)^{pk} \mod N$

2. Prepare Fake Values F
For $i \in [n]$, pick $\rho_i \in \mathbf{Z}_N^*$
$\delta_i \leftarrow (\rho_i)^{pk} \mod N$

3. Mix Sets.
Randomly permute
$\{d_1 \dots d_m, \delta_1 \dots \delta_n\}$
to $\{\beta_1 \dots \beta_{m+n}\}$
Let R be the indices of the d_i
Let F be the indices of the δ_i

$\xrightarrow{\beta_1 \dots \beta_{m+n}}$

4. Evaluation
For $i = 1 \dots m+n$
Evaluate β_i: $s_i = \beta_i^{sk} \mod N$
Encrypt the result s_i:
– Choose random $k_i \in \{0,1\}^{\lambda_1}$
– $c_i = H^{\text{prg}}(k_i) \oplus s_i$
Commit to the keys: $h_i = H(k_i)$

$\xleftarrow{c_1, \dots, c_{m+n}}$

$\xleftarrow{h_1, \dots, h_{m+n}}$

$\xrightarrow{F, \rho_i\ \forall i \in F}$

5. Identify Fake Set F

6. Check Fake Set F
For all $i \in F$:
Verify $\beta_i = (\rho_i)^{pk} \mod N$,
If yes, reveal $k_i\ \forall i \in [F]$.
Else abort.

$\xleftarrow{k_i\ \forall i \in F}$

7. Check Fake Set F
For all $i \in F$,
Verify that $h_i = H(k_i)$
Decrypt $s_i = H^{\text{prg}}(k_i) \oplus c_i$
Verify $(s_i)^{pk} = (\rho_i) \mod N$
Abort if any check fails.

8. Post transaction T_{puzzle}
T_{puzzle} offers 1 bitcoin within timewindow tw_1 under condition "the fulfilling transaction is signed by $\mathcal{T}$ and has preimages of $h_j\ \forall j \in R$".

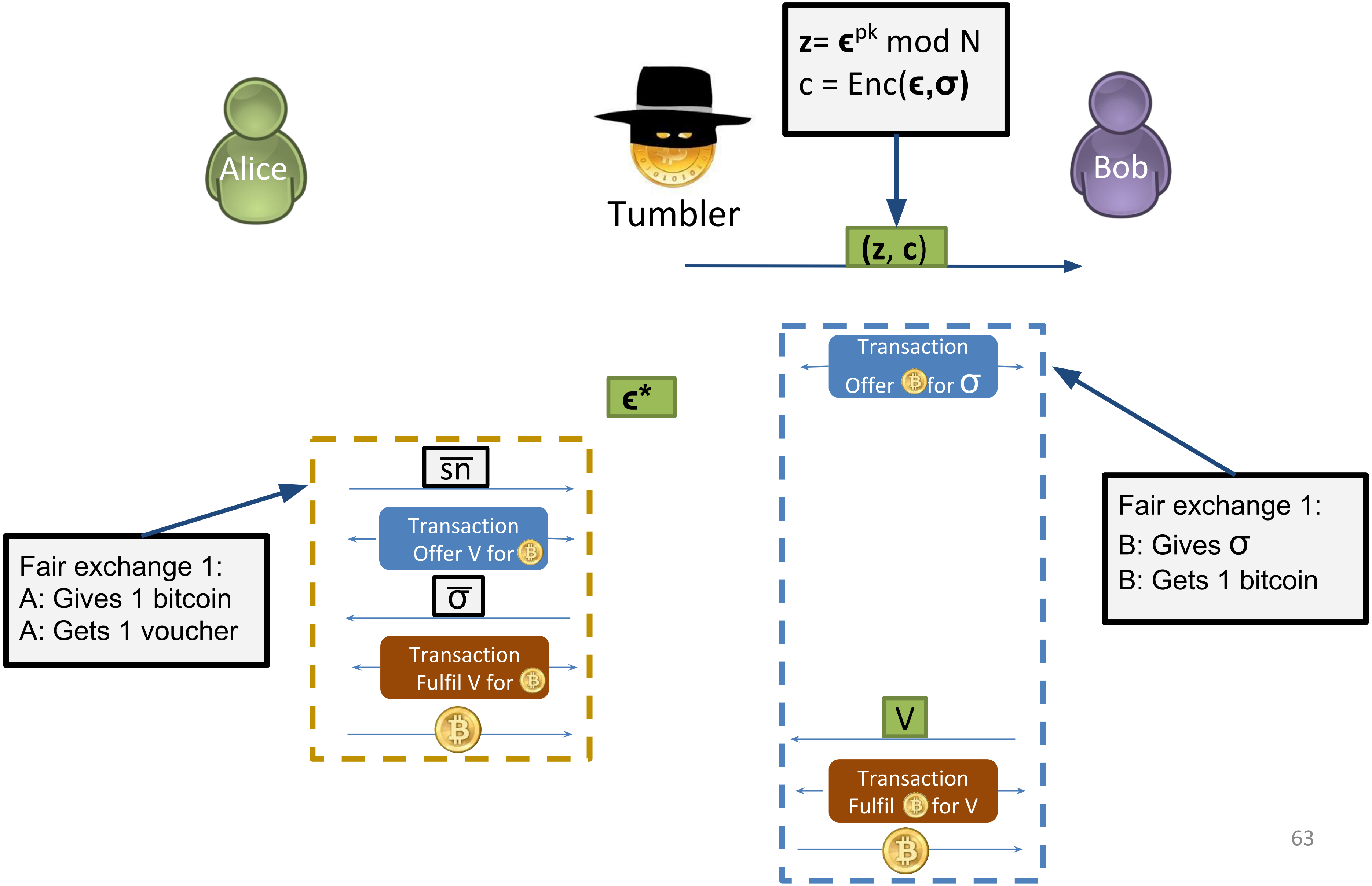
$\xrightarrow{y,\ \ r_j \forall j \in R}$

9. Check β_j unblind to $y\ \forall j \in R$
For all $j \in R$
Verify $\beta_j = y \cdot (r_j)^{pk} \mod N$
If not, abort.

10. Post transaction T_{solve}
T_{solve} contains $k_j \forall j \in R$

11. Obtain Puzzle Solution
For $j \in R$:
Learn k_i from T_{solve}
Decrypt c_j to $s_j = H^{\text{prg}}(k_j) \oplus c_j$
If s_j is s.t. $(s_j)^{pk} = \beta_j \mod N$,
Obtain solution $s_j / r_j \mod N$
which is y^{sk}.

TumbleBit: Paying with RSA-Puzzles



Alice

Tumbler

Bob

$z = \epsilon^{pk} \bmod N$
$c = \mathrm{Enc}(\epsilon, \sigma)$

(z, c)

ϵ^*

$\overline{sn}$

Transaction
Offer V for ₿

$\overline{\sigma}$

Transaction
Fulfil V for ₿

₿

Fair exchange 1:
A: Gives 1 bitcoin
A: Gets 1 voucher

Transaction
Offer ₿ for σ

V

Transaction
Fulfil ₿ for V

₿

Fair exchange 1:
B: Gives σ
B: Gets 1 bitcoin

Bitcoin faces three technology challenges:

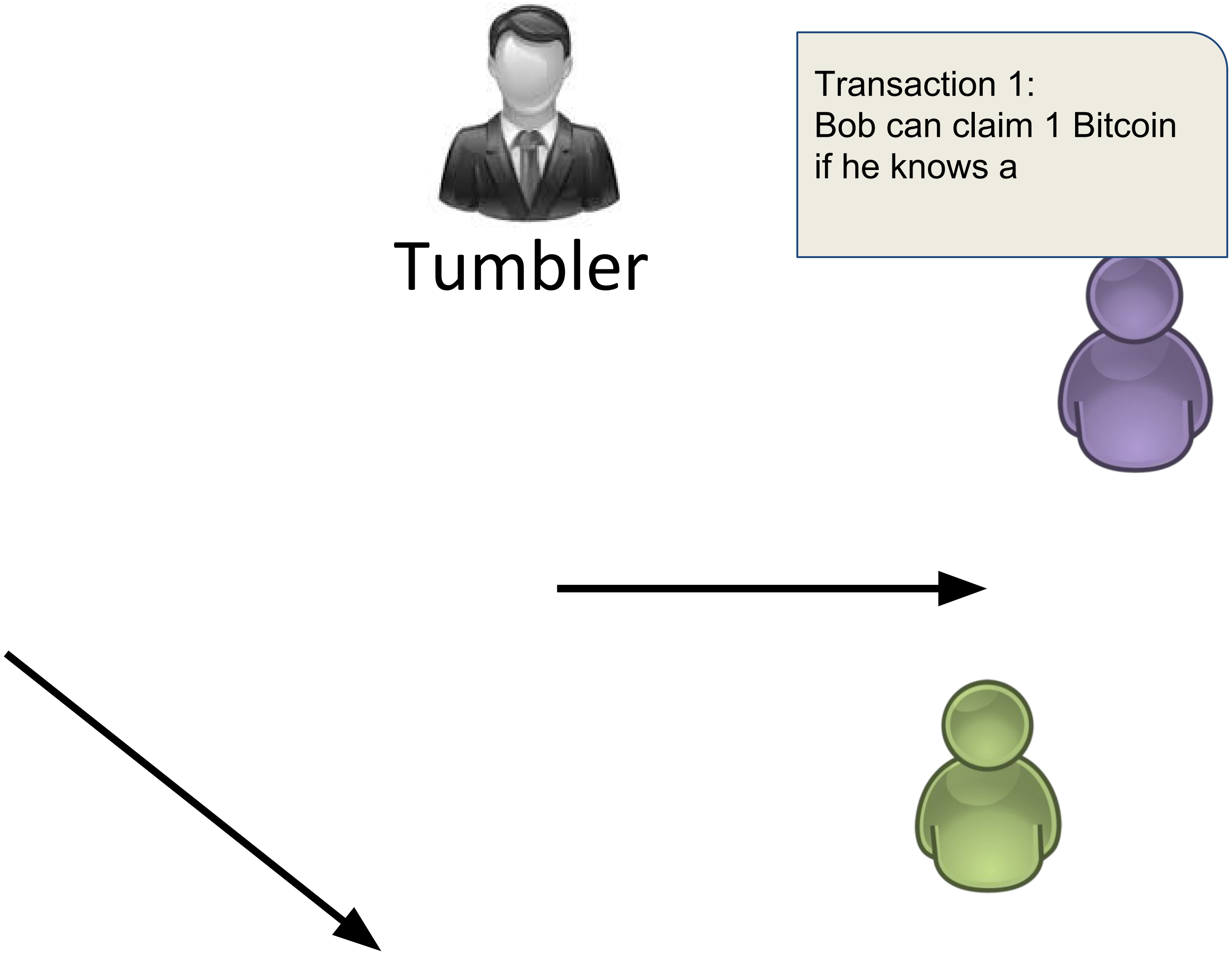
1. **Scaling transaction velocity (speed of payments):**
   - Bitcoin transaction confirmations is ~10 min,
     … occasionally an hour or more.
   - No confirmation = no double spending protection.

2. **Scaling transaction volume (max # of payments):**
   - “Bitcoin achieves 7 transactions/sec maximum throughput
     …[Visa] processes 2000 transaction/sec on average,
     with a peak rate of 56,000 transactions/sec”[1]
   - To compete with mainstream payment processors
     … Bitcoin needs to support much higher transaction volume.
   - Limiting factor here is space in the blockchain.

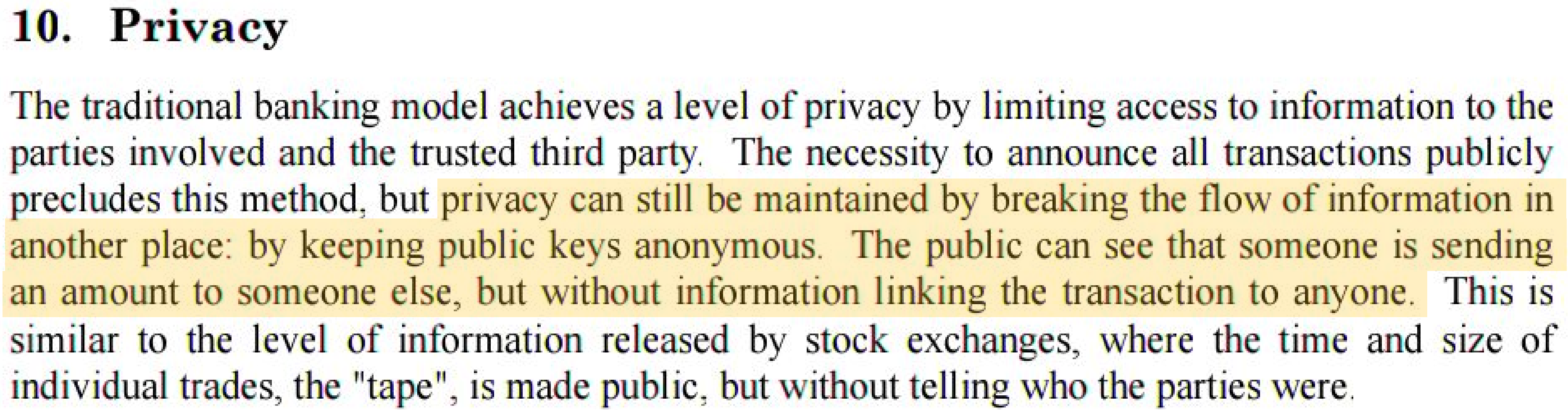
3. **Anonymity and user privacy:**
   - Bitcoin transactions are saved in the blockchain
     … creating an eternal public record of payment history.

[1]: ‘On Scaling Decentralized Blockchains (A Position Paper)’ Croman, et al.

Paying with RSA-Puzzles


Tumbler

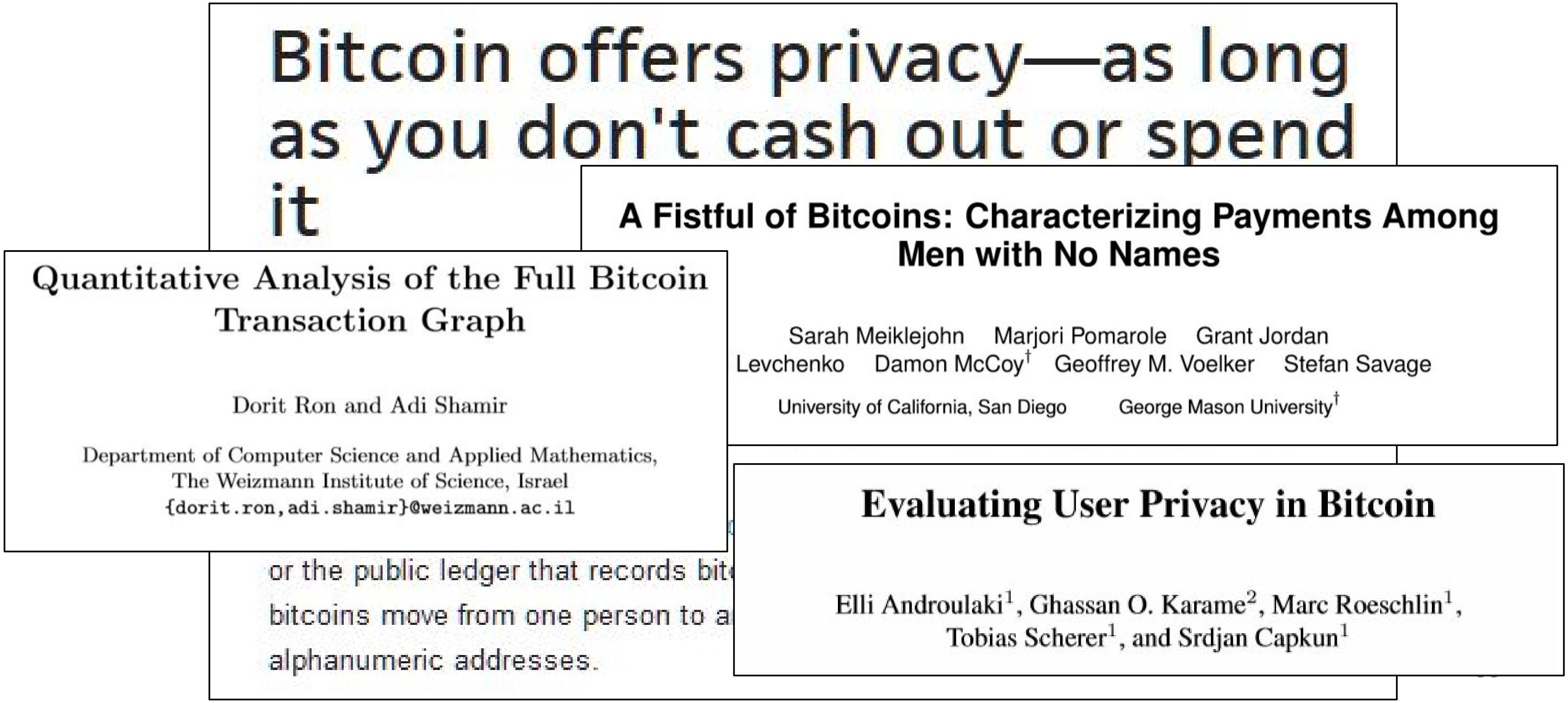
Transaction 1:
Bob can claim 1 Bitcoin
if he knows a



10. Privacy

The traditional banking model achieves a level of privacy by limiting access to information to the parties involved and the trusted third party. The necessity to announce all transactions publicly precludes this method, but privacy can still be maintained by breaking the flow of information in another place: by keeping public keys anonymous. The public can see that someone is sending an amount to someone else, but without information linking the transaction to anyone. This is similar to the level of information released by stock exchanges, where the time and size of individual trades, the "tape", is made public, but without telling who the parties were.

Satoshi Nakamoto, 2008



Bitcoin offers privacy—as long as you don't cash out or spend it

or the public ledger that records bit

bitcoins move from one person to a

alphanumeric addresses.

A Fistful of Bitcoins: Characterizing Payments Among Men with No Names

Sarah Meiklejohn Marjori Pomarole Grant Jordan
Levchenko Damon McCoy[†] Geoffrey M. Voelker Stefan Savage
University of California, San Diego George Mason University[†]

Quantitative Analysis of the Full Bitcoin Transaction Graph

Dorit Ron and Adi Shamir

Department of Computer Science and Applied Mathematics,
The Weizmann Institute of Science, Israel
`{dorit.ron,adi.shamir}@weizmann.ac.il`

Evaluating User Privacy in Bitcoin

Elli Androulaki[1], Ghassan O. Karame[2], Marc Roeschlin[1],
Tobias Scherer[1], and Srdjan Capkun[1]

Introduction

Privacy:

- Bitcoin is not anonymous
- Payment history saved in an eternal public record

Transaction velocity:

- Transactions confirmed on the blockchain
- No confirmation = double spending possible
- Avg confirmation time is ~10 min

Transaction volume: Max # payments

- Bitcoin: 7 Tx/sec max throughput[1]
- Visa: (avg) 2000 Tx/sec[1]
- Visa: (peak) 56,000 Tx/sec[1]
- Limiting factor is space in the blockchain

[1]: 'On Scaling Decentralized Blockchains (A Position Paper)' Croman, et al.

Introduction

Technical challenges facing Bitcoin:

Privacy:

- Bitcoin is not anonymous
- Payment history saved in an eternal public record

Transaction velocity:

- Transactions confirmed on the blockchain
- No confirmation = double spending possible
- Avg confirmation time is ~10 min

Transaction volume: Max # payments

- Bitcoin: 7 Tx/sec max throughput[1]
- Visa: (avg) 2000 Tx/sec[1]
- Visa: (peak) 56,000 Tx/sec[1]
- Limiting factor is space in the blockchain

[1]: ‘On Scaling Decentralized Blockchains (A Position Paper)’ Croman, et al.

TumbleBit: scalability and payment privacy.

✓ **1. Scaling transaction velocity (speed of payments):**

- TumbleBit as a payment hub can make payments in seconds.

✓ **2. Scaling transaction volume (max # of payments):**

- Payment hubs allow many payments to one party to be aggregated into two on-blockchain transactions.
- These payments don’t need to be stored or validated on the blockchain.

✓ **3. Anonymity and payment privacy:**

- TumbleBit provides payment privacy via unlinkability.

In this talk I am only going to tell you about how TumbleBit provides trustless payment privacy.