TumbleBit:
An Untrusted Bitcoin-Compatible Anonymous Payment Hub

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

BOSTON UNIVERSITY

Scaling Bitcoin Milan 2016

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:

TumbleBit can be used as a classic Bitcoin tumbler:

- k-anonymity within a mix,
- 4 transactions confirmed in 2 blocks (~20mins)

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.

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:

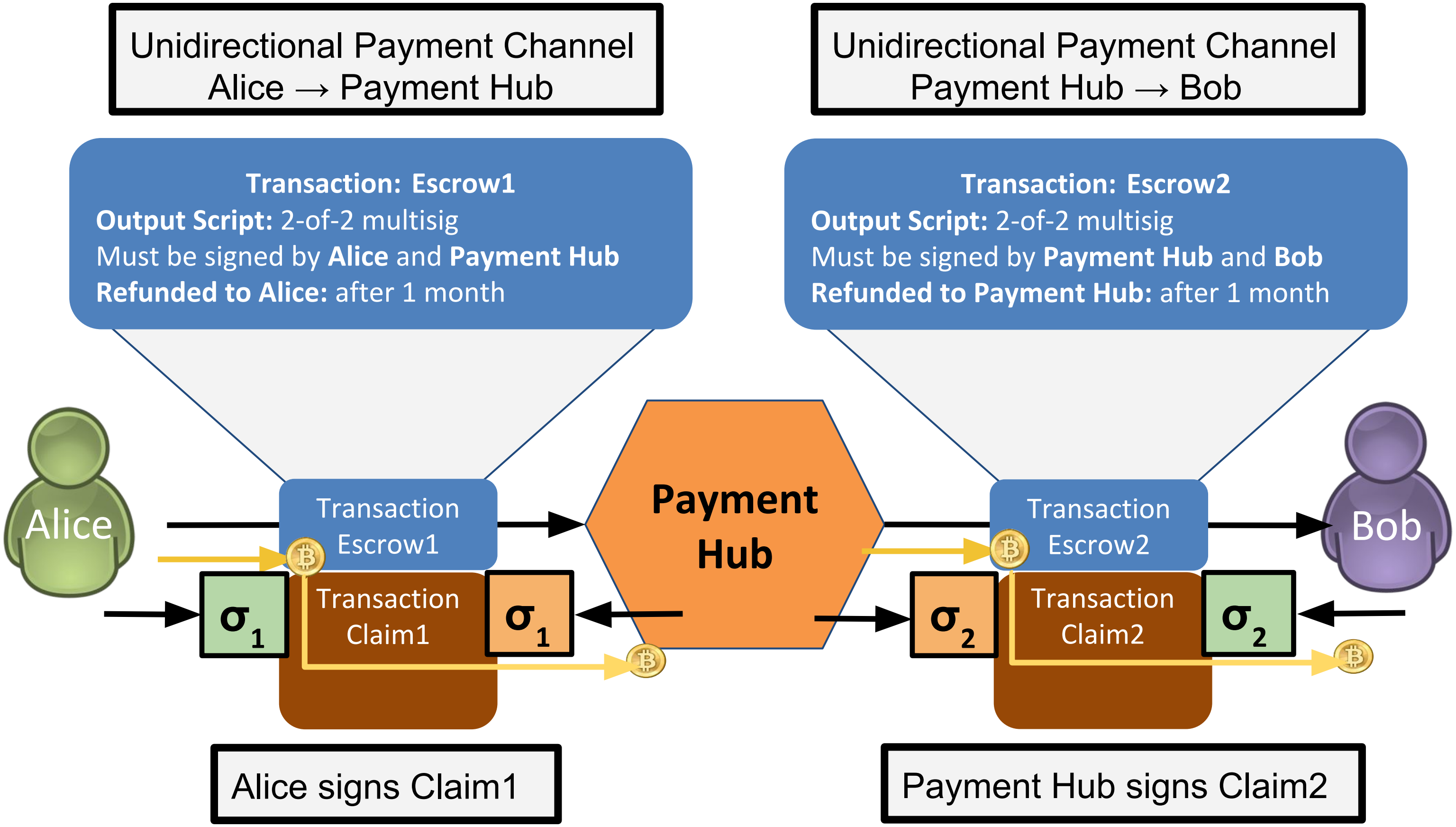
When used as a payment hub, TumbleBit helps scale Bitcoin's transaction velocity (faster payments), and transaction volume (higher maximum payments).

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

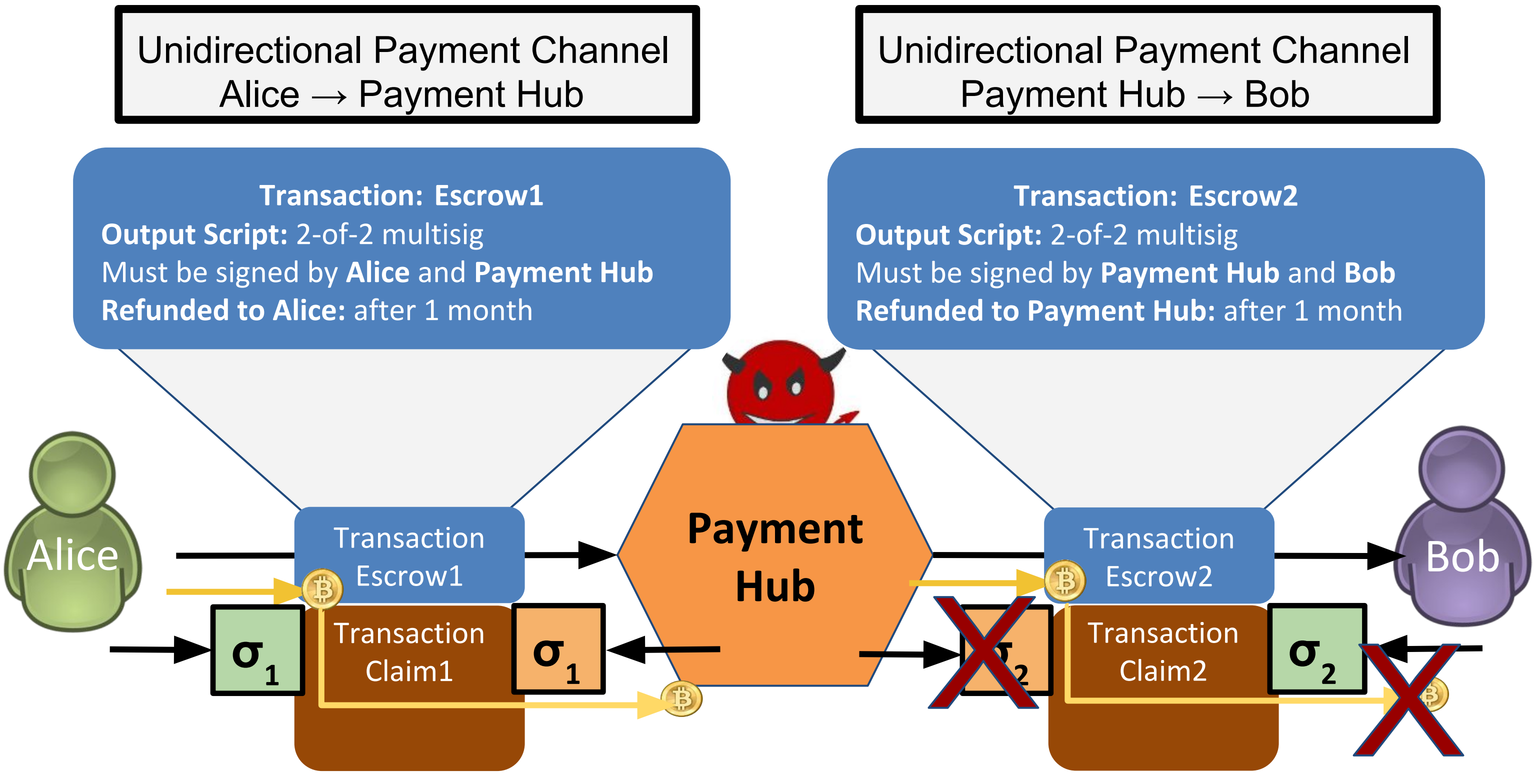
A payment hub: routes payment channels.



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

Background: Payment Hub

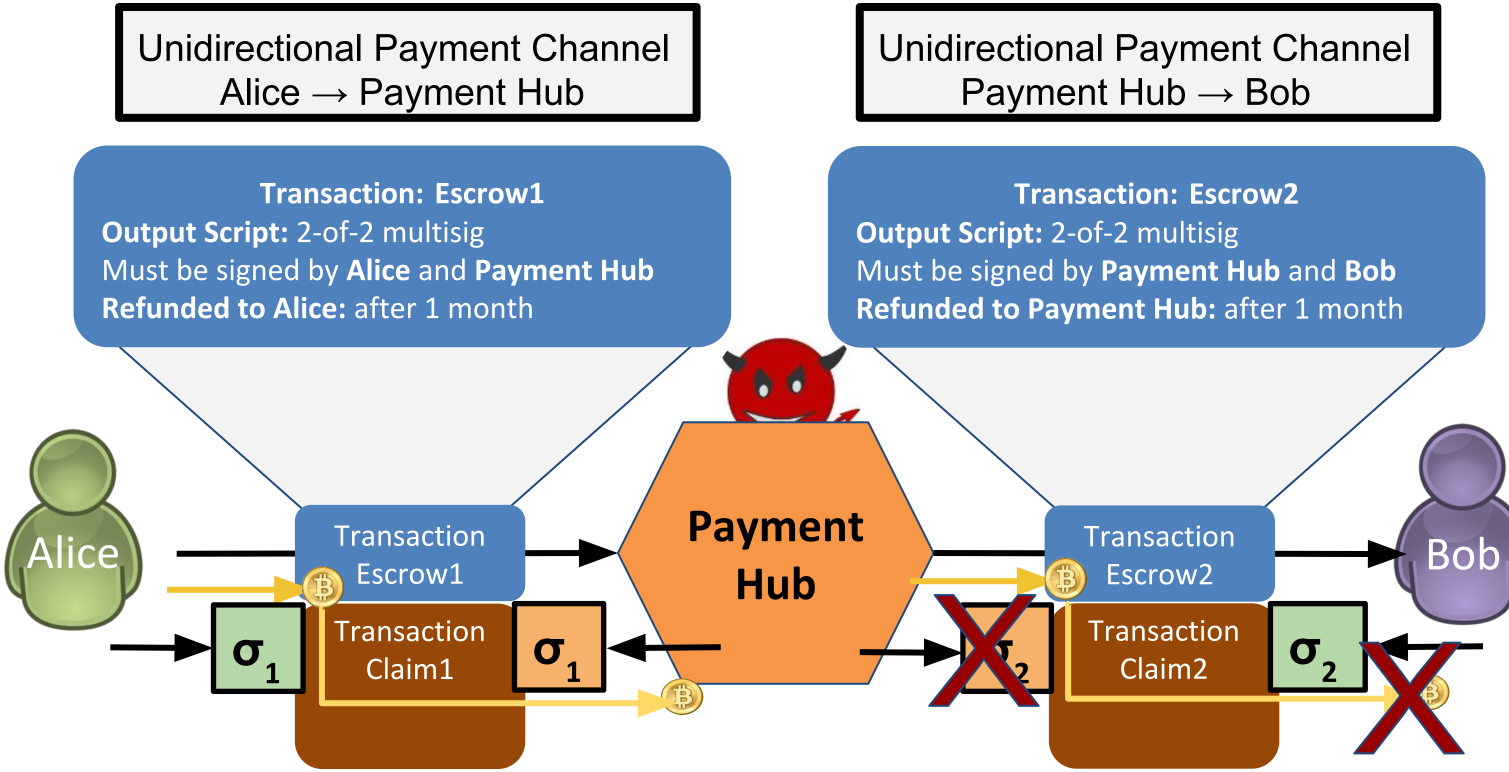
A payment hub: routes payment channels.



...But what if the hub is malicious, and takes Alice's bitcoin and doesn't pay Bob?

Background: Payment Hub

A payment hub: routes payment channels.



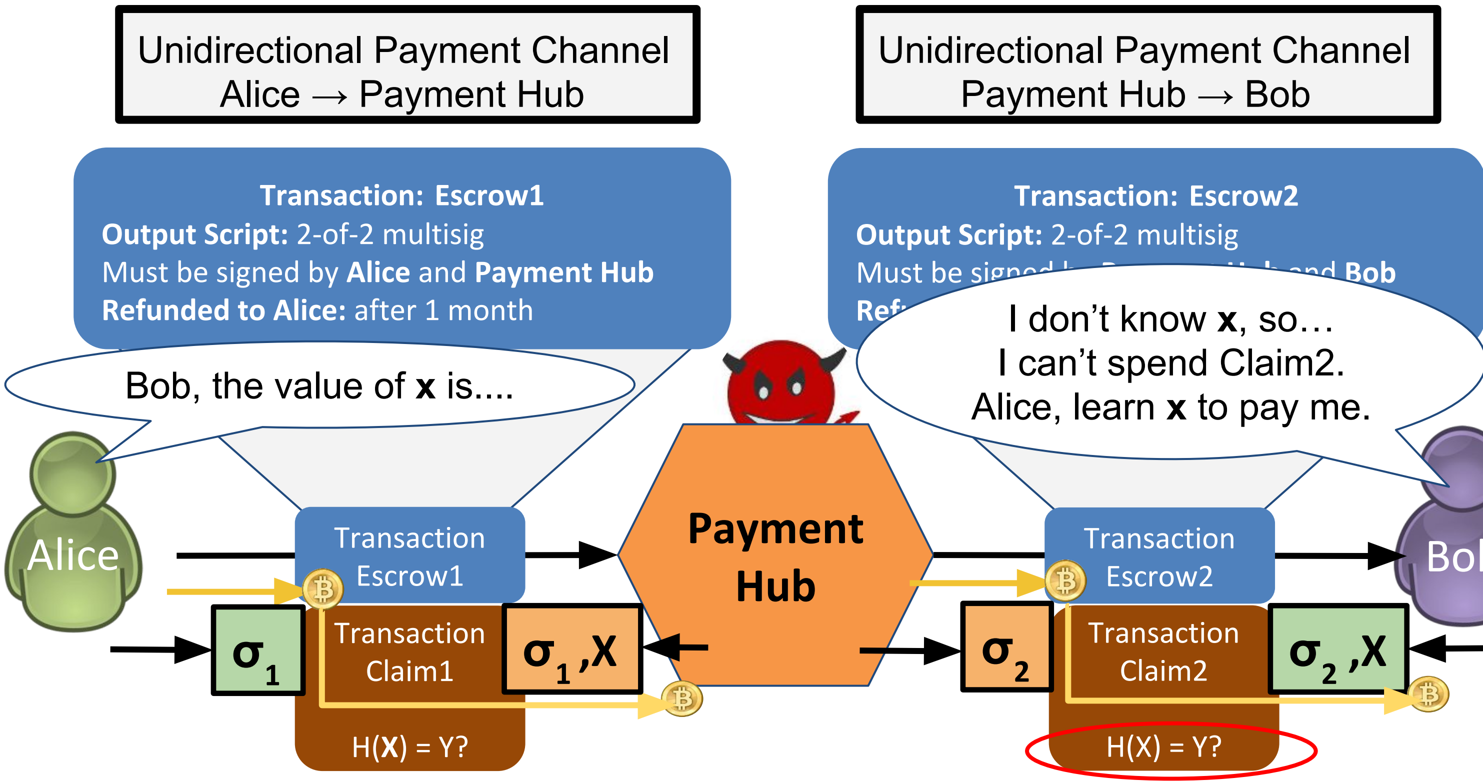
...But what if the hub is malicious,

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

Hash locks provide this property.

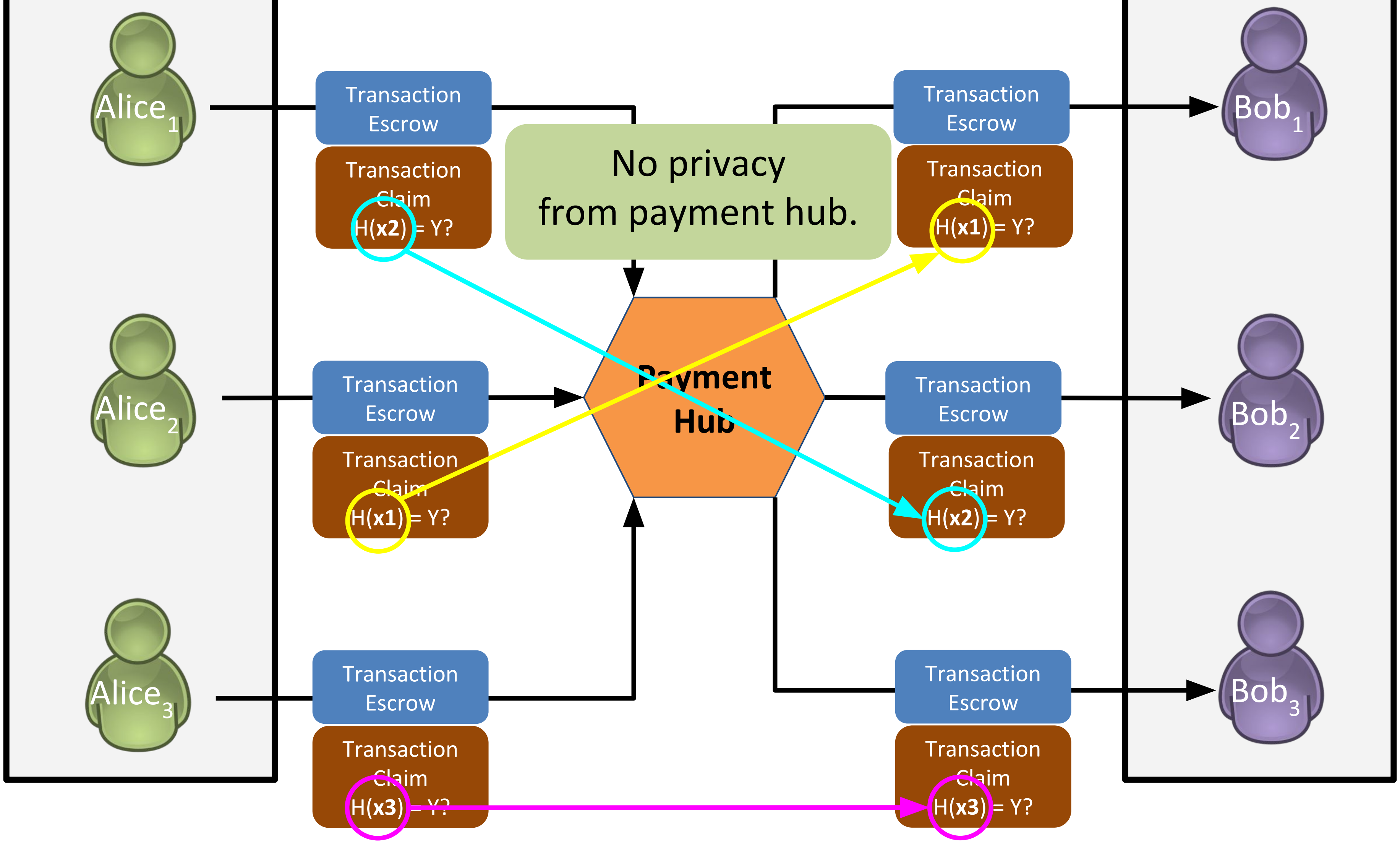
Background: Payment Hub

A payment hub: routes payment channels.

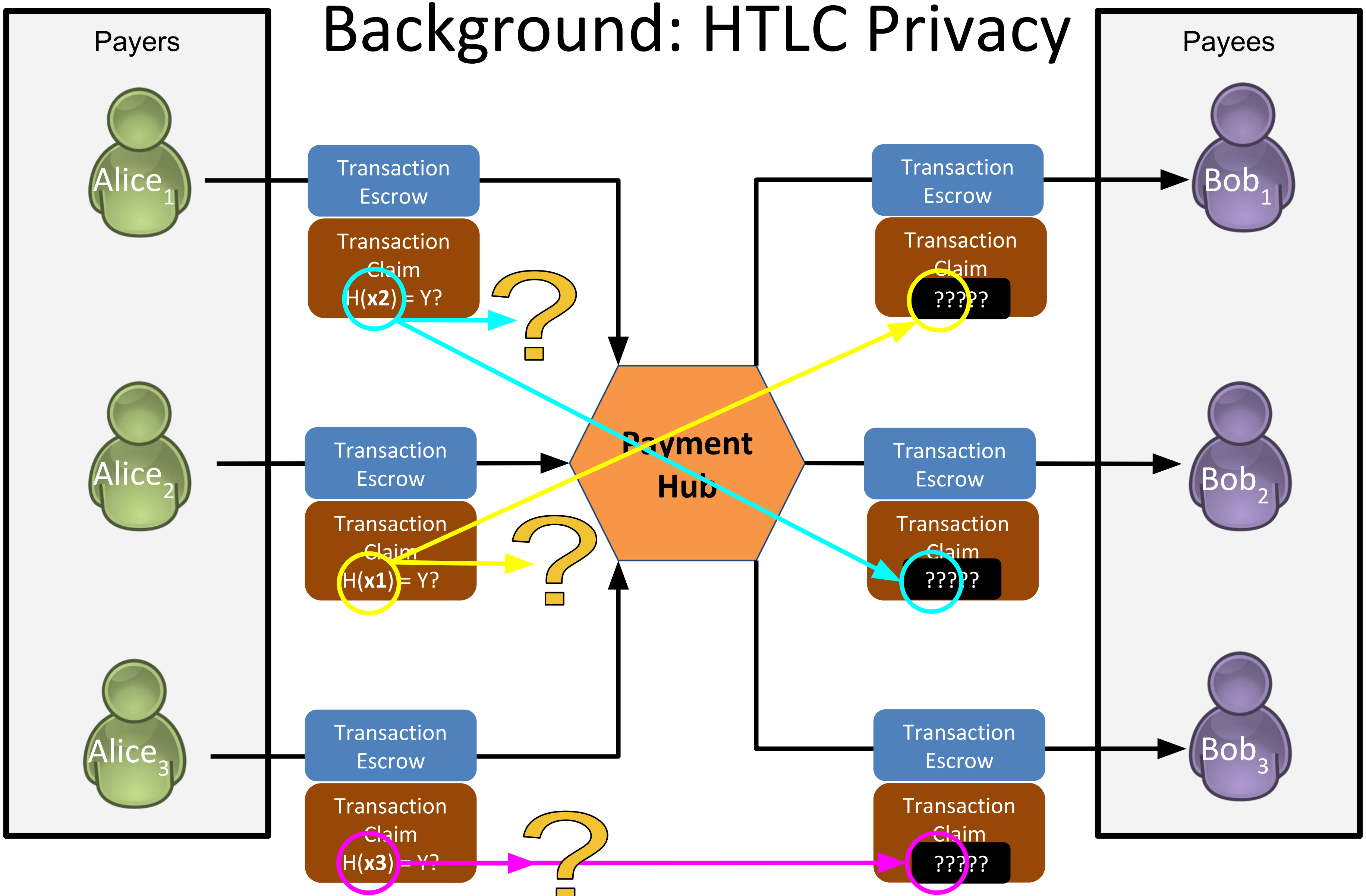


Thus, using hash locked transactions or HTLCs a payment hub can prevent theft, however this is provides no privacy against the payment hub.

Background: HTLC Privacy



Background: HTLC Privacy



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.

Background: RSA Puzzles

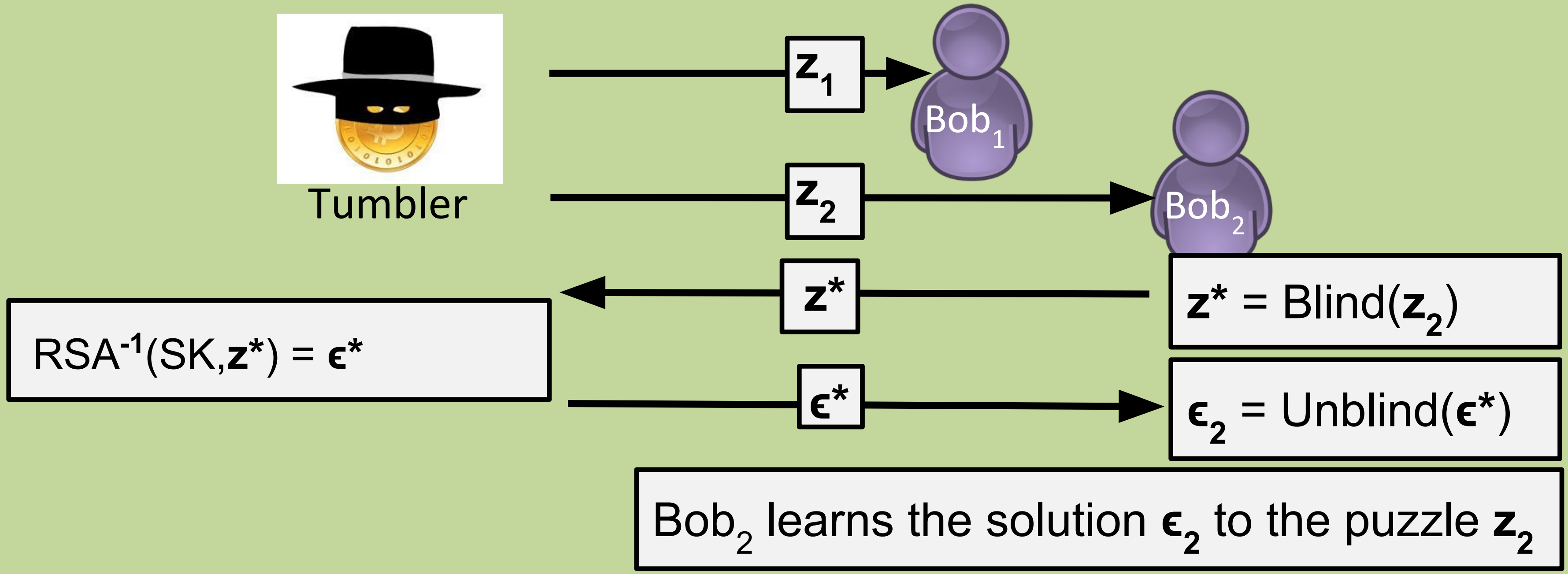
- An RSA Puzzle is just a "textbook RSA encryption" of some value $\boldsymbol{\epsilon}$:

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

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

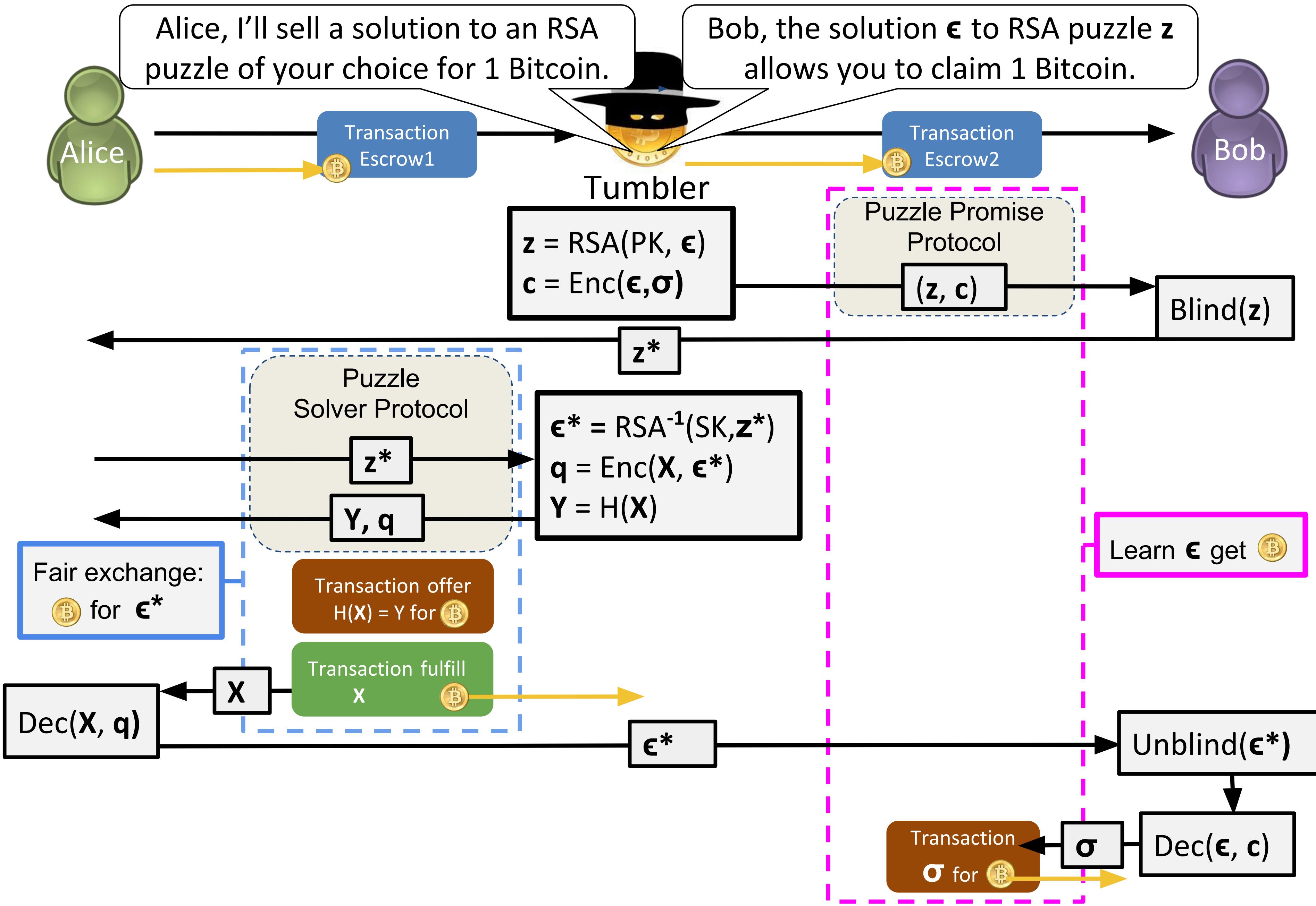
$$\mathrm{RSA}^{-1}(\mathrm{SK}, \mathbf{z}) = \mathrm{RSA}^{-1}(\mathrm{SK}, \mathrm{RSA}(\mathrm{PK}, \boldsymbol{\epsilon})) = \boldsymbol{\epsilon}$$

RSA blinding can be used to blind RSA puzzles

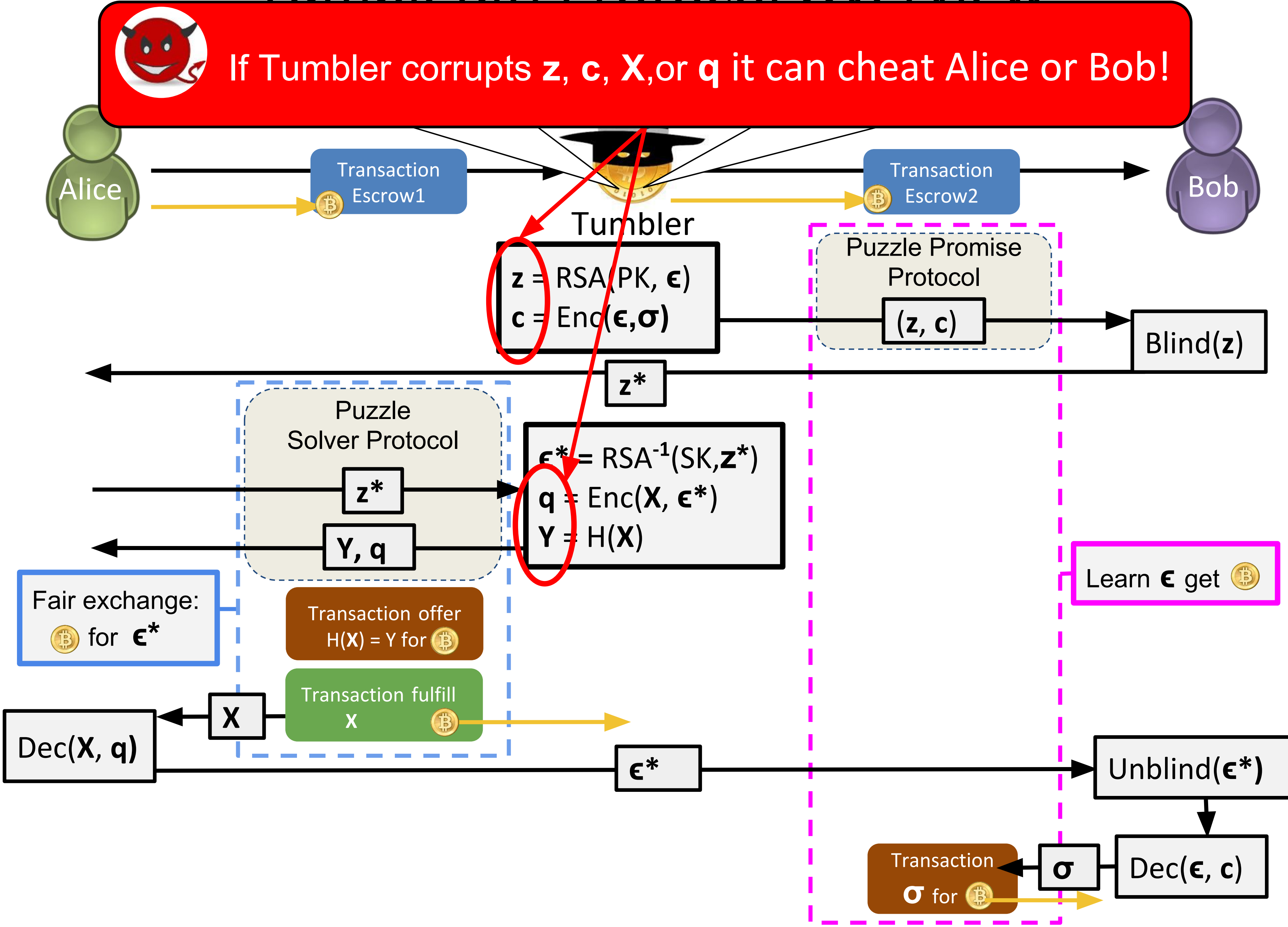


Tumbler can not link the blinded RSA puzzle it solves $\mathbf{z}^*$ to any of the RSA puzzles it issued ($\mathbf{z}_1$, $\mathbf{z}_2$).

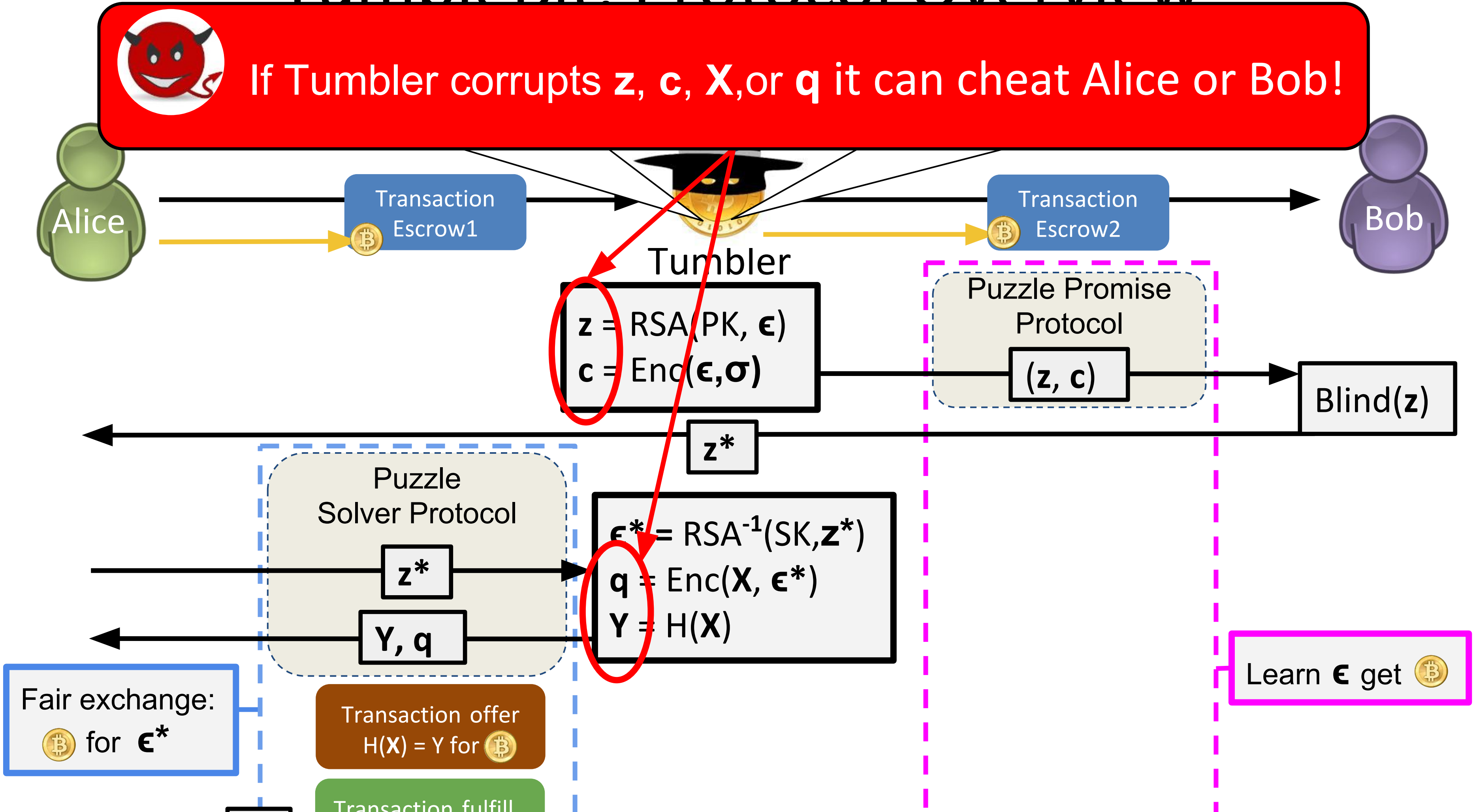
TumbleBit: Protocol Overview



TumbleBit: Protocol Overview



TumbleBit: Protocol Overview

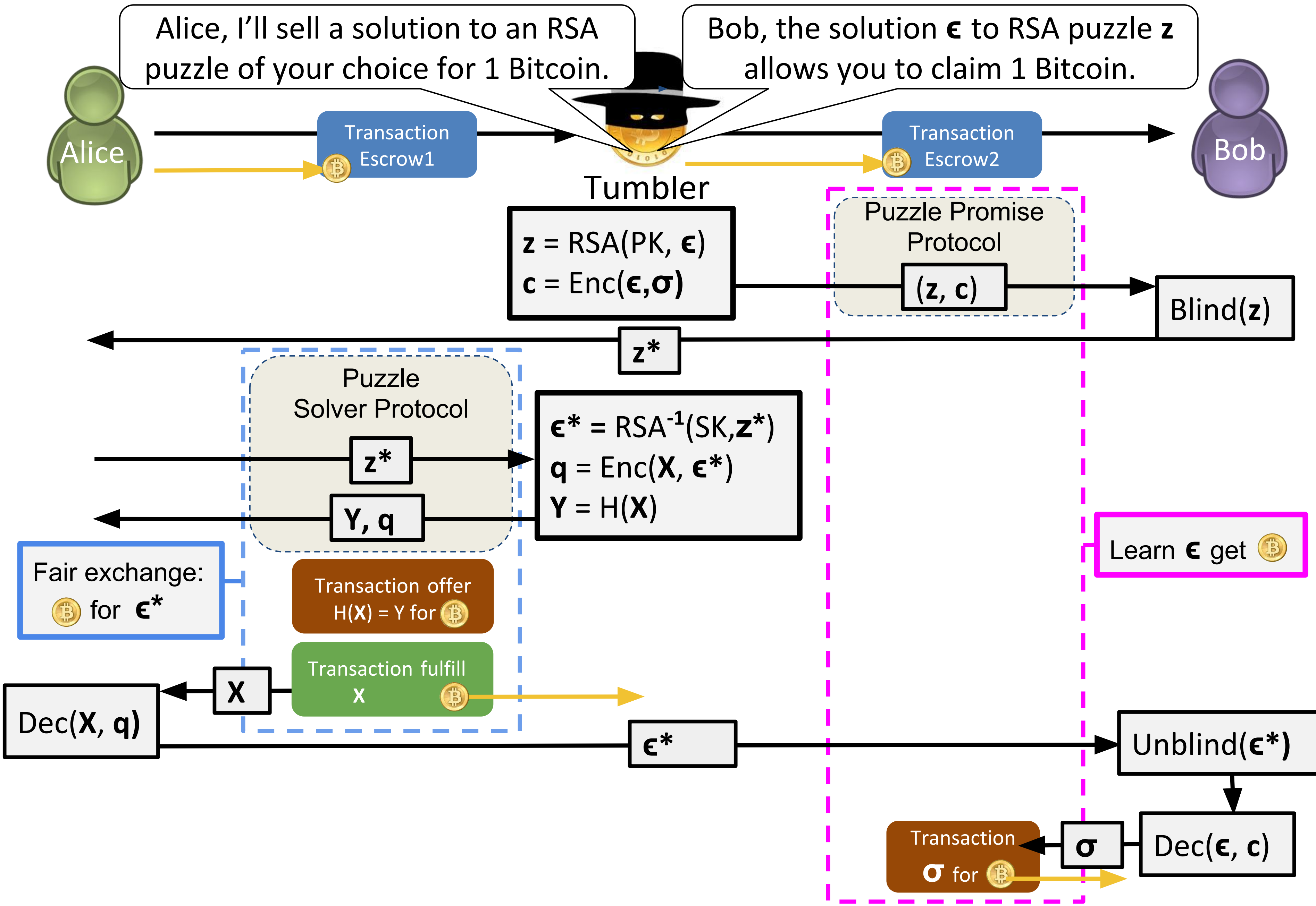


TumbleBit prevents this via two protocols:

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

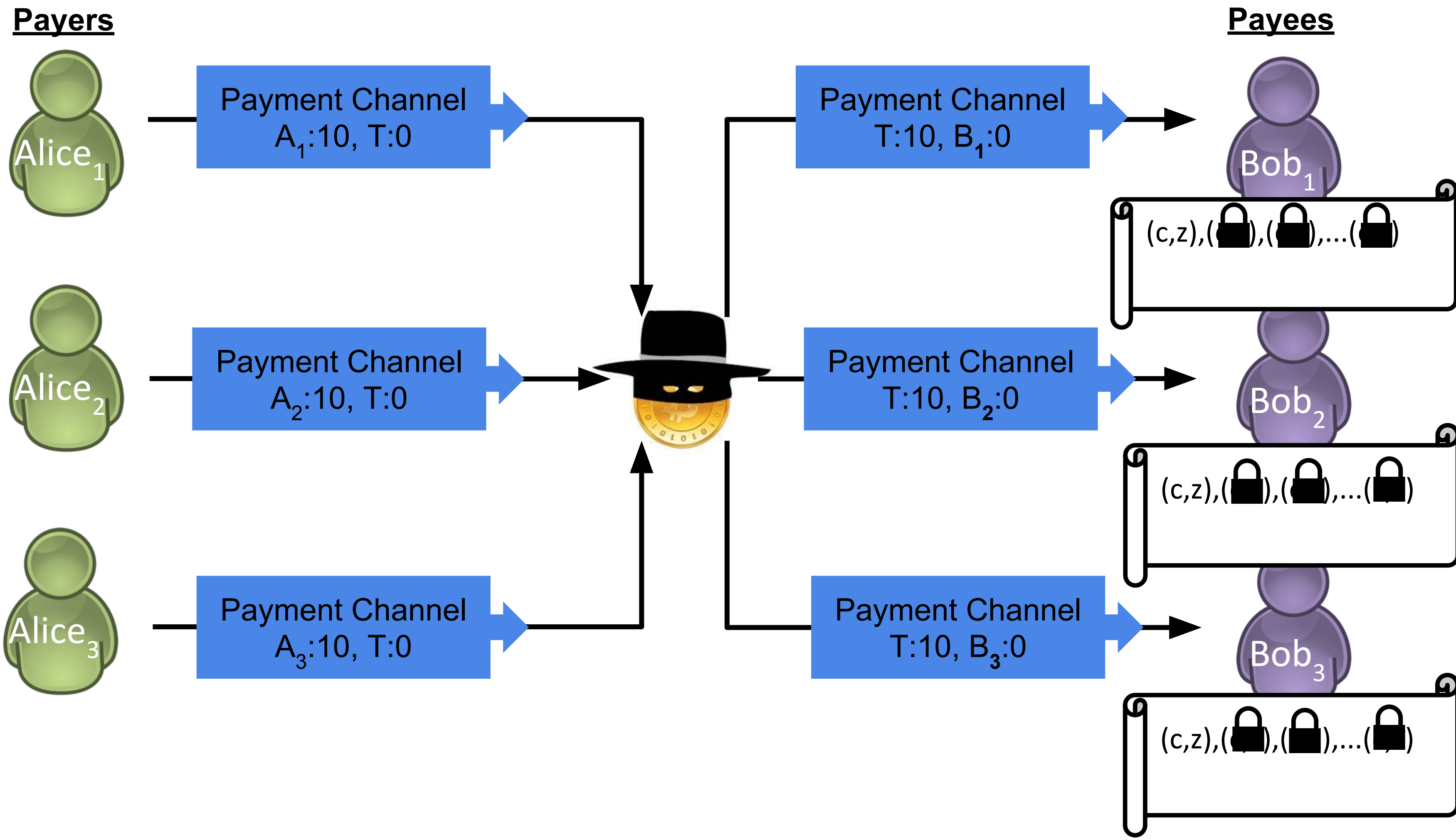
Puzzle-Promise-Protocol:
Tumbler convinces Bob that the solution to RSA puzzle **z** is a value **ϵ** which allows him learn **σ**.

TumbleBit: Protocol Overview



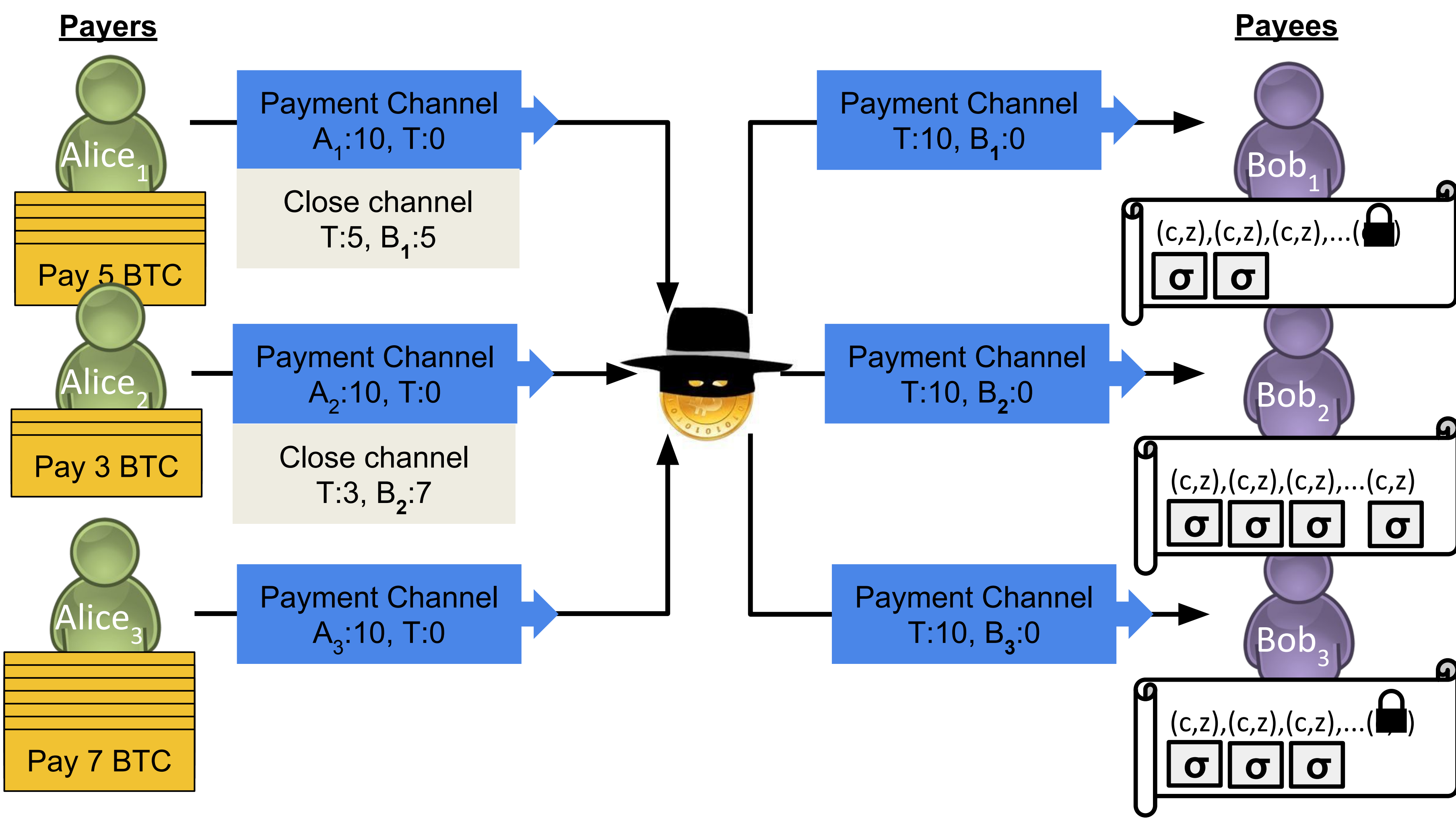
TumbleBit: Phases

1. **Escrow Phase:** All payment channels setup.



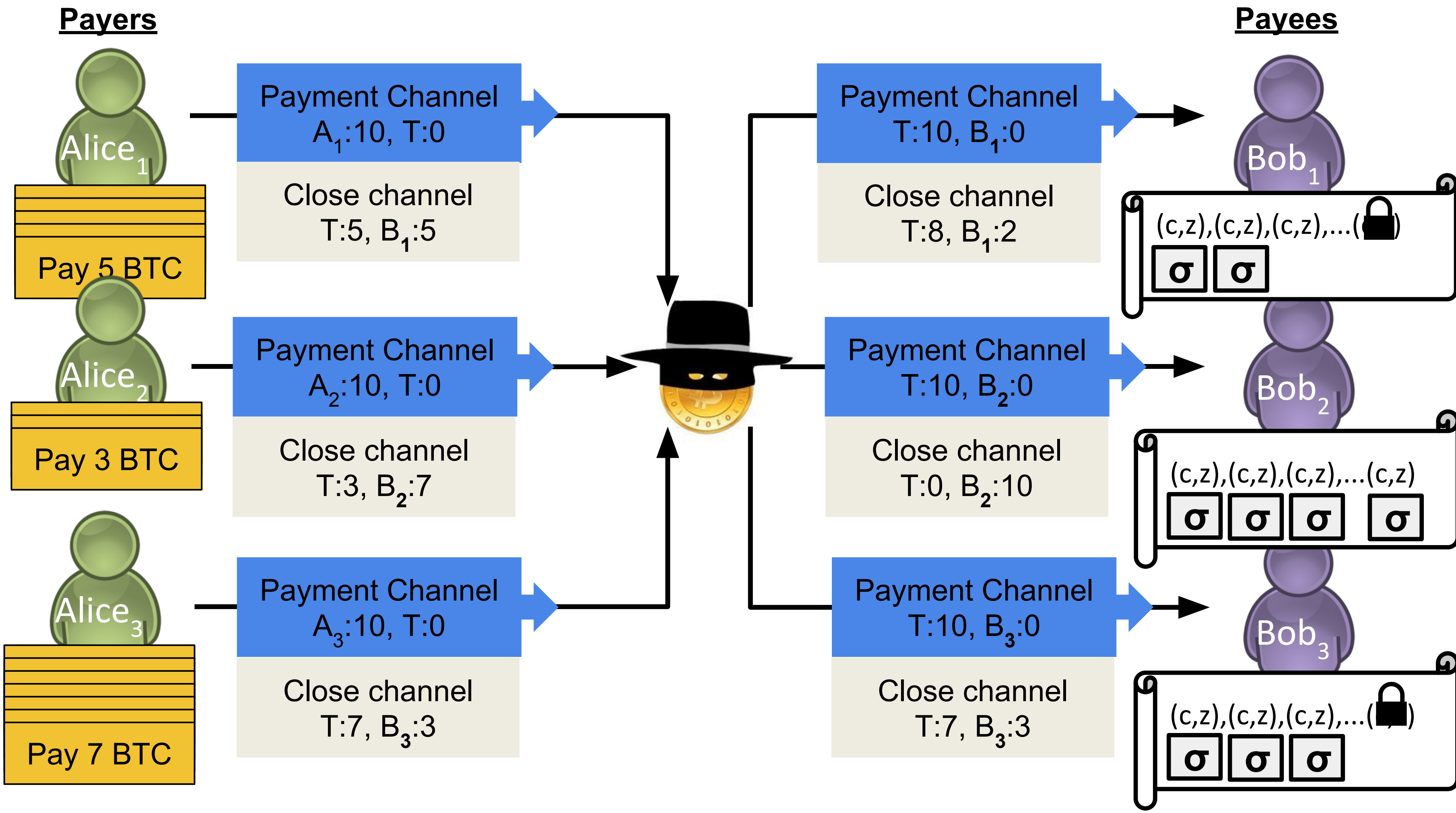
TumbleBit: Phases

1. **Escrow Phase:** All payment channels setup.
2. **Payments Phase (~1 month):** Payers make payments.



TumbleBit: Phases

1. **Escrow Phase:** All payment channels setup.
2. **Payments Phase (~1 month):** Payers make payments.
3. **Cashout Phase:** Payers and payees close their payment channels.

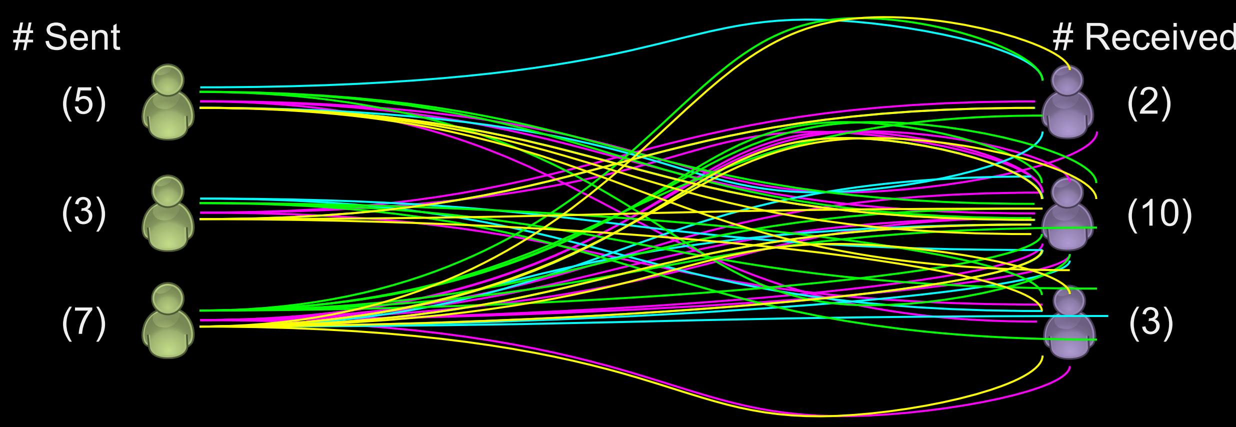


TumbleBit: Phases

Privacy offered the TumbleBit Payment Hub

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

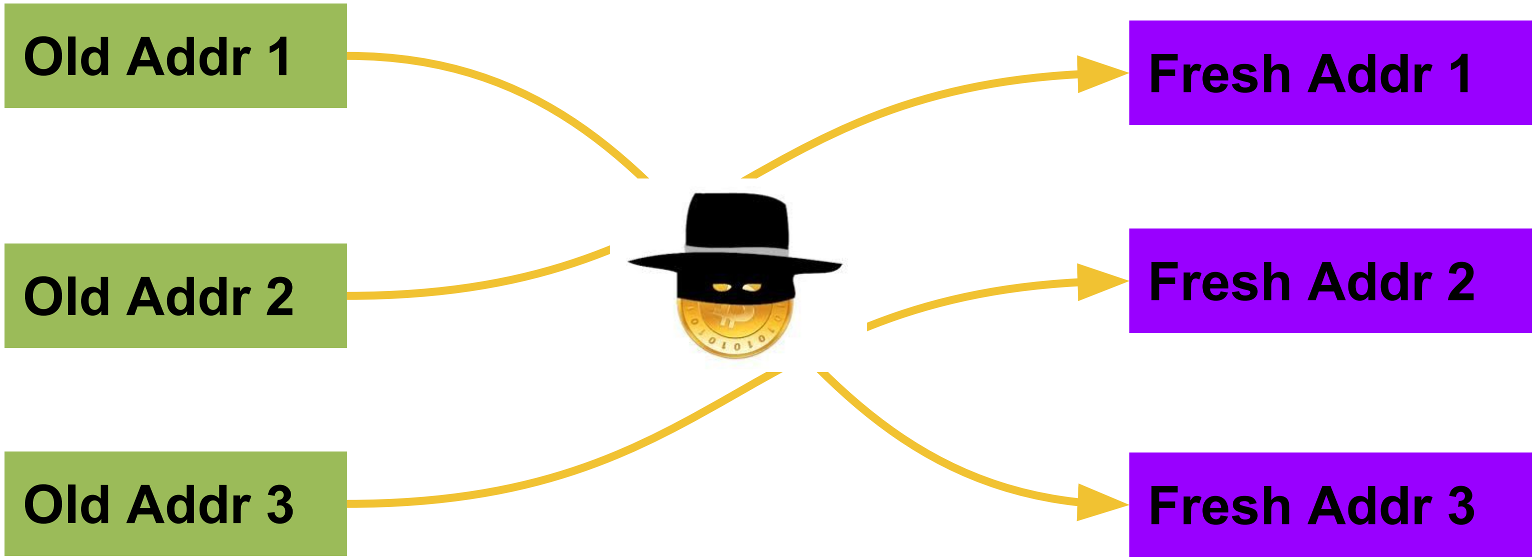
Unlinkability def:
All interaction graphs compatible with the tumblers view are equally likely.



TumbleBit: Classic Tumbler

TumbleBit can also be a classic tumbler:

Allows users to privately move bitcoins to an unlinked fresh address.

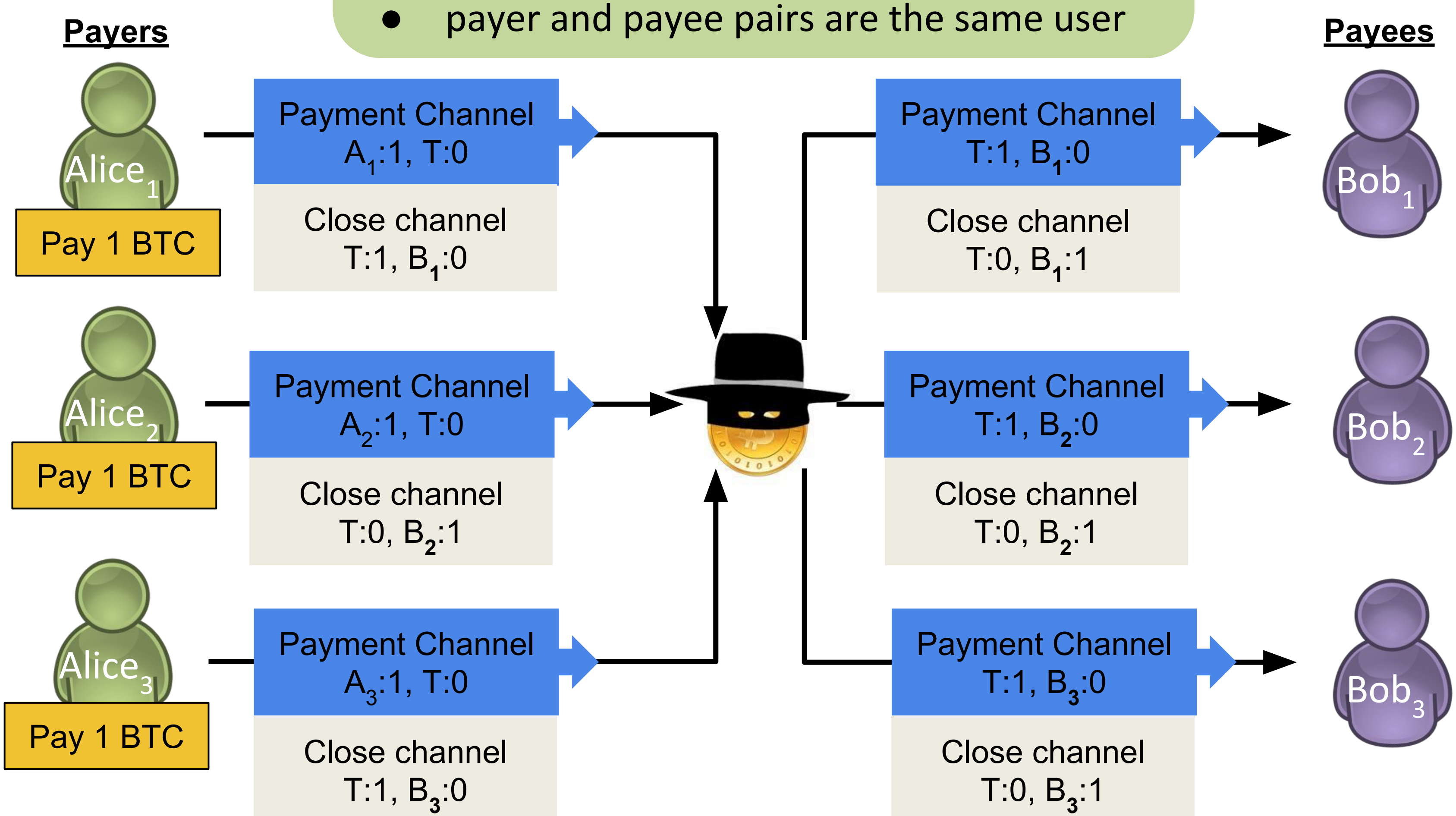


This is also sometimes known as a mixing service or mix.

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.
- payer and payee pairs are the same user



Provides k-anonymity:
Where k = # of payers = # of payee.

Compared to other Tumblers

Vulnerable to DoS & Sybil Attacks

CoinJoin:

Limited Anonymity

CoinShuffle

Vulnerable to bitcoin theft

mixcoin

True Anonymous Cryptocurrency

Blindcoin:

CoinSwap

Simple. Fast. Exchange

Intermediary breaks anonymity

TumbleBit

Mixing takes hours

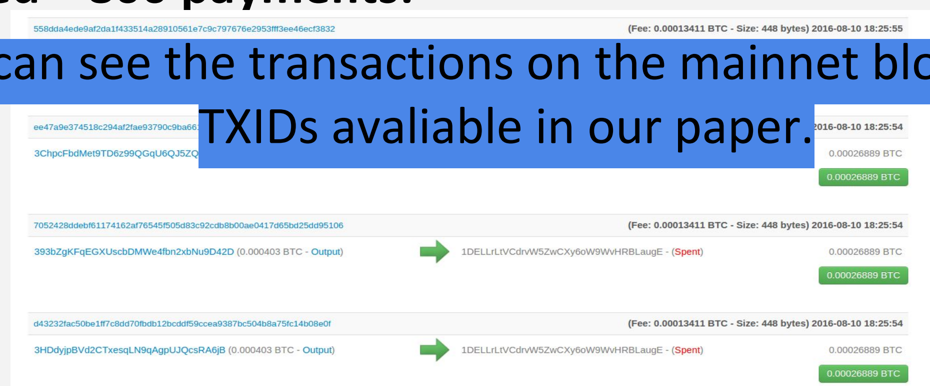
Xim

TumbleBit: Implementation

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

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

We “tumbled” 800 payments:



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 use TOR)

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):

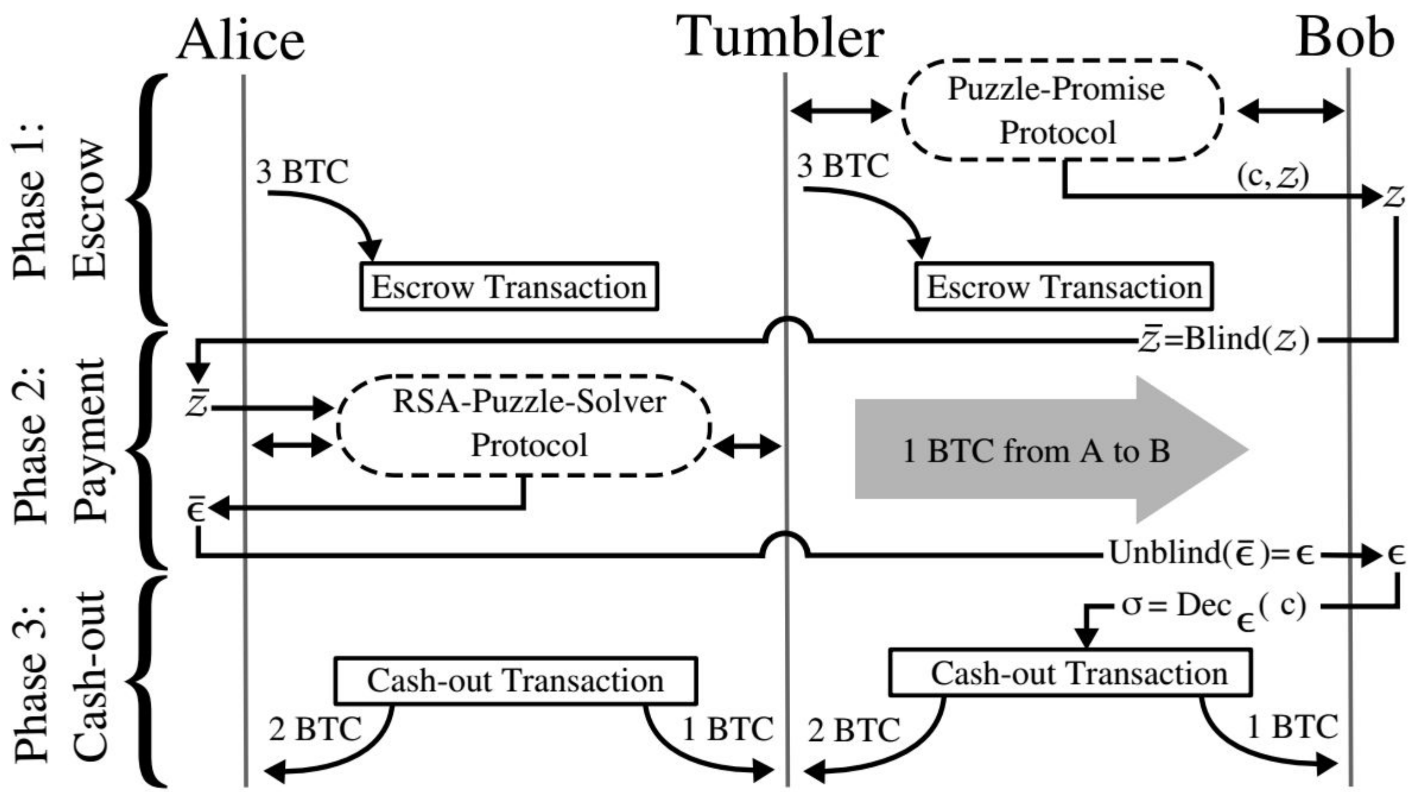
- Our code runs on Bitcoin's mainnet blockchain.
- We have published our code on github.
- ...and we working to improve it and make TumbleBit easy and safe to use.

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 **ϵ*** if and only if Alice pays.

z*

Tumbler

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)

(q1,Y1),(q2,Y2),(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-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 **ϵ*** if and only if Alice pays.

z*

Alice

Tumbler

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)

(q1,Y1),(q2,Y2),(q3,Y3),...

3. Reveal
by sendi

Probability(Tumbler successfully cheats) = (m+n choose m) = ~1/(2^{80})
m = # of real puzzles = 15
n = # of fake puzzles = 285

5. Check
values "I
correctly computed.

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

(R1, R2, … Rm)

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

This is why the protocol is hard,
otherwise Tumbler could convince Bob
by just sending (**c,z,ϵ,σ**) and let Bob check.

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 $\boldsymbol{\sigma}$ under a key $\boldsymbol{\epsilon}$ i.e Dec(**ϵ,c**) = $\boldsymbol{\sigma}$
2. **AND** the key $\boldsymbol{\epsilon}$ is the solution to the RSA-puzzle **z.**

The protocol should never: allow Bob to learn a valid $\boldsymbol{\sigma}$ (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**:
$\boldsymbol{\sigma}$**i** = Sign(**Bi**)
zi = RSA^{-1}(SK,**ϵi**), **ci** = Enc(**ϵi,σi**)

This is why the protocol is hard,
otherwise Tumbler could convince Bob
by just sending (**c,z,ϵ,σ**) and let Bob check.

3. B: reveals transactions.

T1,~~T2~~,~~T3~~,T4,~~T5~~,T6

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: 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 = 42

n = # of invalid transactions = 42

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.