TumbleBit:
An Untrusted Bitcoin-Compatible Anonymous Payment Hub

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

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.

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

Transaction:

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

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.

**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

**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, and takes Alice’s bitcoin and doesn’t pay Bob?
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

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 Bob: after 1 month

---

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

No privacy from payment hub.

Transaction Claim $H(x) = Y$?
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 $\epsilon$:
  \[ \text{RSA}(PK, \epsilon) = z \]

- Only the party that knows SK can solve RSA puzzles:
  \[ \text{RSA}^{-1}(SK, z) = \text{RSA}^{-1}(SK, \text{RSA}(PK, \epsilon)) = \epsilon \]

RSA blinding can be used to blind RSA puzzles

\[ \epsilon^* = \text{Unblind}(\epsilon^*) \]

Bob$_2$ learns the solution $\epsilon^*_2$ to the puzzle $z^*_2$

Tumbler can not link the blinded RSA puzzle it solves $z^*$ to any of the RSA puzzles it issued ($z^*_1$, $z^*_2$).
TumbleBit: Protocol Overview

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

Bob, the solution $\epsilon$ to RSA puzzle $z$ allows you to claim 1 Bitcoin.

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

Bob, the solution $\epsilon$ to RSA puzzle $z$ allows you to claim 1 Bitcoin.

Fair exchange: $\sigma$ for $\epsilon^*$

Transaction offer $H(X) = Y$ for $\epsilon^*$

Transaction fulfill $X$

Dec($X, q)$

Unblind($\epsilon^*$)

Transaction $\sigma$ for $\epsilon^*$

Dec($\epsilon, c$)
TumbleBit: Protocol Overview

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

Fair exchange: for $\epsilon^*$

Transaction offer $H(X) = Y$ for

Transaction fulfill $X$

Dec($X$, $q$)

Puzzle Solver Protocol

$z^* = RSA^{-1}(SK, z^*)$

$q = Enc(X, \epsilon^*)$

$Y = H(X)$

$Y, q$

Dec($X$, $q$)

Unblind($\epsilon^*$)

Puzzle Promise Protocol

$(z, c)$

Blind($z$)

Transaction Escrow1

Tumbler

Transaction Escrow2

$puzzle-solver-protocol:
Tumbler convinces Alice the preimage $X$ where $Hash(X) = Y$ will allow her to learn $\epsilon^*$.

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 $\text{Hash}(X) = Y$ will allow her to learn $\epsilon^*$.  

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

---

If Tumbler corrupts $z$, $c$, $X$, or $q$ it can cheat Alice or Bob!
TumbleBit: Protocol Overview

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

Bob, the solution $\epsilon$ to RSA puzzle $z$ allows you to claim 1 Bitcoin.

**Puzzle Solver Protocol**

$z = RSA(PK, \epsilon)$

$c = Enc(\epsilon, \sigma)$

$(z, c)$

Bob, the solution $\epsilon$ to RSA puzzle $z$ allows you to claim 1 Bitcoin.

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

**Transaction Escrow1**

**Transaction Escrow2**

Blind($z$)

Learn $\epsilon$ get $\text{Bitcoin}$

Dec($X, q$)

$X$

Dec($\epsilon, c$)

Unblind($\epsilon*$)

**Fair exchange:**

for $\epsilon^*$

**Transaction offer**

$H(X) = Y$ for $\text{Bitcoin}$

**Transaction fulfill**

$X$

**Dec($X, q$)**

$\epsilon^*$

$\text{Bitcoin}$

$\text{Bitcoin}$
TumbleBit: Phases

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

```plaintext
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.
```

- **Payers**
  - Alice₁
    - Payment Channel \( A₁:10, T:0 \)
  - Alice₂
    - Payment Channel \( A₂:10, T:0 \)
  - Alice₃
    - Payment Channel \( A₃:10, T:0 \)

- **Payees**
  - Bob₁
    - Payment Channel \( T:10, B₁:0 \)
  - Bob₂
    - Payment Channel \( T:10, B₂:0 \)
  - Bob₃
    - Payment Channel \( T:10, B₃:0 \)

\( (c,z),(c,z),(c,z),... \)
\( \sigma \)
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.

### Payers
- Alice₁
  - Pay 5 BTC
- Alice₂
  - Pay 3 BTC
- Alice₃
  - Pay 7 BTC

### Payment Channels
- Payment Channel $A_1:10$, $T:0$
  - Close channel $T:5$, $B_1:5$
- Payment Channel $A_2:10$, $T:0$
  - Close channel $T:3$, $B_2:7$
- Payment Channel $A_3:10$, $T:0$
  - Close channel $T:7$, $B_3:3$

### Payees
- Bob₁
- Bob₂
- Bob₃

### Close Channels
- Close channel $T:0$, $B_2:10$
- Close channel $T:0$, $B_2:10$
- Close channel $T:0$, $B_2:10$
- Close channel $T:7$, $B_3:3$
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.

<table>
<thead>
<tr>
<th># Sent</th>
<th># Received</th>
</tr>
</thead>
<tbody>
<tr>
<td>(5)</td>
<td>(2)</td>
</tr>
<tr>
<td>(3)</td>
<td>(10)</td>
</tr>
<tr>
<td>(7)</td>
<td>(3)</td>
</tr>
</tbody>
</table>
TumbleBit: Classic Tumbler

TumbleBit can also be a classic tumbler:

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

Old Addr 1 → Fresh Addr 1
Old Addr 2 → Fresh Addr 2
Old Addr 3 → Fresh Addr 3

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 payee.
- 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
  - CoinShuffle

- **Limited Anonymity**
  - TumbleBit
  - Xim

- **Vulnerable to bitcoin theft**
  - Blindcoin

- **Intermediary breaks anonymity**
  - Mixcoin
  - Coinswap
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 Github.

We “tumbled” 800 payments:

- You can see the transactions on the mainnet blockchain. TXIDs available 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


Ask questions on twitter: @Ethan_Heilman
**TumbleBit: Puzzle-Solver-Protocol**

**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 $\epsilon^*$ if and only if Alice pays.

---

1. **Makes $m$ real puzzles:**
   - for $i$ in $m$: $D_i = \text{Blind}(z^*, R_i)$

2. **Solves/encrypts:**
   - for $i$ in $m+n$:
     - $\epsilon_i = \text{RSA}^{-1}(SK, B_i)$
     - $q_i = \text{Enc}(X_i, S_i)$
     - $Y_i = H(X_i)$

3. **Reveals fake puzzles by sending solutions.**

4. **Reveals $X_i$ of fake puzzles.**

5. **Checks fake puzzles values** “$H(X) = Y$” correctly computed.

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

7. **Decrypts $q_i$’s learns $\epsilon^*$**

---

**If Tumbler computes any $(q_i, \epsilon_i, Y_i)$ of the real puzzles correctly Alice learns $\epsilon^*$, thus to cheat Alice, Tumbler must corrupt all the real and none of the fake puzzles.**
TumbleBit: Puzzle-Solver-Protocol

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 $\epsilon^*$ if and only if Alice pays.

Probability(Tumbler successfully cheats) = \binom{m+n}{m} = \sim 1/2^{80}

$m = \# \text{ of real puzzles} = 15$

$n = \# \text{ of fake puzzles} = 285$

If Tumbler computes any $(q_i, \epsilon_i, Y_i)$ of the real puzzles correctly Alice learns $\epsilon^*$, 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 \(\sigma\) under a key \(\epsilon\) i.e \(\text{Dec}(\epsilon, c) = \sigma\)
2. AND the key \(\epsilon\) is the solution to the RSA-puzzle \(z\).

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

This is why the protocol is hard, otherwise Tumbler could convince Bob by just sending \((c, z, \epsilon, \sigma)\) and let Bob check.
TumbleBit: Puzzle-Promise-Protocol

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 \( \sigma \)**: For Bi in B:
   
   \[ \sigma_i = \text{Sign}(Bi) \]
   
   \[ z_i = \text{RSA}^{-1}(SK, \epsilon_i), \ c_i = \text{Enc}(\epsilon_i, \sigma_i) \]

3. **B reveals transactions.**

4. **T Reveals**: \( \epsilon_i \) for invalid transactions.

5. **B checks**: invalid transactions \( \sigma_i \) are correctly computed.

6. **Bob and Tumbler run “quotient protocol” ensuring that**: If Bob learns \( \epsilon_1 \), Bob can use that knowledge to learn \( \epsilon_4, \epsilon_6 \).

\[ (\epsilon_4/\epsilon_1 \mod N, \epsilon_6/\epsilon_4 \mod N) \]

If Tumbler computes any \((\epsilon_i, \sigma_i)\) of the valid transactions correctly Bob learns a \( \sigma \)/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 \(\sigma\) under a key \(\epsilon\) i.e \(\text{Dec}(\epsilon, C) = \sigma\)
2. AND the key \(\epsilon\) is the solution to the RSA-puzzle \(z\).

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

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

\[ B = H(T1), H(T2), H(T3), H(T4), H(T5), H(T6) \]

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

2. T Signs & Encrypts \(\sigma\):
   for \(B_i\) in \(B\):
   \[ \sigma_i = \text{Sign}(B_i) \]
   \[ z_i = \text{RSA}^{-1}(SK, \epsilon_i) \]
   \[ C_i = \text{Enc}(\epsilon_i, \sigma_i) \]

3. B checks: invalid transactions \(\sigma_i\) are correctly computed.

4. T Reveals: \(\epsilon_i\) for invalid transactions.

5. Bob and Tumbler run “quotient protocol” ensuring that:
   if Bob learns \(\epsilon_1\), Bob can use that knowledge to learn \(\epsilon_4, \epsilon_6\).
   \((\epsilon_4/\epsilon_1 \mod N, \epsilon_6/\epsilon_4 \mod N)\)

Probability(Tumbler successfully cheats) = \(\binom{m+n}{m}\) = \(\sim 1/(2^{80})\)

\(m = \#\) of valid transactions = 42
\(n = \#\) of invalid transactions = 42

If Tumbler computes any \((\epsilon_i, \sigma_i)\) of the valid transactions correctly Bob learns a \(\sigma\)/gets paid, thus to cheat Bob, Tumbler must all corrupt all the valid and none of the invalid transactions.