Astrape — Anonymous Payment Channels With Boring Cryptography

Yuhao Dong

November 20, 2020

University of Waterloo

Introduction

The problem with blockchains

Blockchain-based cryptocurrencies promise to revolutionarize money.

- Decentralized
- Censorship-resistant
- Completely permissionless

Unfortunately, blockchains aren't a very good money ledger:

- Lack of privacy — can render censorship resistance meaningless
- Very poor transaction throughput

The problem with blockchains

One way of fixing this is by improving blockchains

- Privacy: Zcash, Monero, etc
    - Terrible performance!
- Scalability: sharding, dPoS, etc
    - Complexity/centralization

A better way: **payment channels**

Payment channels

What is a payment channel?

- Alice and Bob each lock \$50 into a vault.
- When transactions happen between Alice and Bob, they privately sign a statement saying which of the \$100 belongs to Alice, and which to Bob.
- Vault can be opened by a signed statement from both Alice and Bob declaring the correct split. But this statement can be overridden with newer statements within a timeout period.

Payment channel networks are mesh networks of users with payment channels open to their neighbors. Transactions are passed through multiple intermediaries.

- Horizontal scalability!

Atomic multi-hop transactions

Payment channel networks require an **atomic multi-hop transaction** construction.

- Pay Alice so that she pays Bob so that he pays Carol
- Ensure that money cannot be stolen by intermediaries

Most common construction: **hash time-lock contract** (HTLC).

- \$100 payable to:
  - Bob if he can find p such that $H(p) = x$ within t hours
  - Alice otherwise

Thus, Alice can randomly generate p, x tell Carol the answer to the puzzle, then send Bob an HTLC. Bob will send Carol an analogous HTLC. Carol solves the puzzle to claim the money, letting Bob claim the money from Alice.

Problems with HTLC

The biggest problem with HTLC is **poor privacy**

- All intermediaries receive coins with the same puzzle.
- Different hops know they're on the same transaction.
- Can lead to deanonymization!

Existing work on fixing HTLC does exist:

- Specialized protocols for special blockchains (Bolt for ZCash, etc)
- Fulgor (Malavolta et al. 2017) — expensive, off-chain zero-knowledge proofs, but compatible with most blockchains
- Tumblebit (Goldberg et al. 2017) — custom construction based on RSA
- AMHL (Malavolta et al. 2019) — on-chain linear homomorphic encryption

Why not existing solutions?

No anonymous PCN construction exists with only HTLC's cryptography

- **Black-box** signature & hash

All are tightly coupled to the mathematics of specific constructions

- You can't "swap out" RSA with e.g. SPHINCS in Tumblebit

Astrape: our construction

Astrape

Astrape is a novel anonymous PCN construction that uses the same tools as HTLC — generic hash functions and signatures.

Three objectives:

- **Relationship anonymity**: Given two simultaneous payments of the same value with paths sharing an honest intermediary, an attacker cannot know which individual transactions belong to which payment pay even if all other intermediaries are compromised.
- **Balance security**: No honest user ever loses money involuntarily.
- **Wormhole resistance**: Attackers cannot make a payment "skip" intended intermediaries to deprive them of fees.

"Onion-routing-like" anonymity, similar to Tumblebit and AMHL

Notation and framework

We ignore other aspects of a PCN and focus on atomic multi-hop transactions.

- U_0 wishes to pay U_n through intermediaries $U_1, \ldots, U_{i-1}$.
- We denote the **coin** that U_i sends to U_{i+1} as $U_i \to U_{i+1}$
- Each coin has a **lock** — a boolean function representing a puzzle that the recipient must solve. For example, we might represent an HTLC as:

$$\begin{aligned}\mathrm{HTLC}[x, t, A, B] \; ::= \; & (\pi, \zeta) \mapsto \\ & (H(\pi) = x \wedge \mathrm{Signed}_B(\zeta)) \vee (\mathrm{Timeout}[t] \wedge \mathrm{Signed}_A(\sigma))\end{aligned}$$

Multi-hop HTLC

We first construct a "strawman" construction that doesn't work — multi-hop HTLC.

- The sender U_0 samples random values $(r_1, \ldots, r_n)$ and calculates $s_i = H(r_i \oplus r_{i+1} \oplus \cdots \oplus r_n)$.
- U_0 then sends $(r_i, s_i, [s_{i+1}])$ to U_i
- Each coin $U_i \to U_{i+1}$ is locked by HTLC$[s_{i+1}, t, U_i, U_{i+1}]$

The recipient U_n can solve the HTLC on s_n. This enables U_{n-1} to solve its puzzle and so on and so forth:

$$H^{-1}(s_i) = r_i \oplus H^{-1}(s_{i+1})$$

Multi-hop HTLC

Multi-hop HTLC does have **relationship anonymity**

- Intuitively, because given a random-oracle hash function, s_i and s_j are random-looking and cannot be correlated

Unfortunately, it has no **balance security**!

- Malicious U_0 can give some intermediary U_i a false r_i.
- When $U_i \to U_{i+1}$ is spent, U_i finds that he cannot spend $U_{i-1} \to U_i$.
- U_n will get paid with U_i's money instead of U_0's!

"Bad state" attack

Fulgor (Malavolta et al. 2017) is simply multi-hop HTLC plus a zero-knowledge proof that U_0 gives each hop that U_0 isn't lying. We can do better.

Fixing multi-hop HTLC

Our central insight: *corrupt senders need no privacy*.

- If the attacker corrupts the sender he already broke all privacy guarantees
- Sender *already knows the whole path* in our model

Thus, we can compose multi-hop HTLC with a **non-anonymous** mechanism

- that's only triggered when the sender is corrupt
- and doesn't leak information unless used

This isn't particularly hard to do with "boring" crypto.

Constructing fraud proofs

After generating the multi-hop HTLC parameters, U_0 generates n values x_i recursively:

$$x_i = H(r_i||s_i||s_{i+1}||o_i||x_{i+1})$$
$$x_n = H(o_n)$$

where o_i is a random nonce.

The intuition here is that x_i commits to all the information the sender would give to all hops U_j where $j \geq i$.

x_i is then given to U_i.

Constructing fraud proofs

Let's consider what happens if U_0 attempts to defraud U_i by giving it the wrong r_i. This generates the following **fraud proof** $\{k_{i+1}, r_i, s_i, s_{i+1}, o_i\}$ where:

$$H(k_{i+1}) = s_i$$
$$H(r_i \oplus (k_{i+1}) \neq s_i$$
$$H(r_i||s_i||s_{i+1}||o_i||x_{i+1}) = x_i$$

that can be verified by anybody with x_i, like U_i.

(k_{i+1} would be the HTLC solution $H^{-1}(s_i)$ that $U_i \rightarrow U_{i+1}$ was spent with.

Constructing fraud proofs

Since x_i commits to *all* multi-hop HTLC initialization states "rightwards" of U_i, U_i, in cooperation with U_{i-1}, can also produce a fraud proof that U_{i-2} can verify using x_{i-2}. This is a set of simply λ-bit values k_{i+1}, r_{i-1}, s_{i-1}, r_i, s_i,s_{i+1}, x_i, x_{i+1} where:

$$
\begin{aligned}
H(k_{i+1}) &= s_{i+1} \\
H(r_i \oplus k_{i+1}) &\neq s_i \\
H(r_i||s_i||s_{i+1}||o_i||x_{i+1}) &= x_i \\
H(r_{i-1}||s_{i-1}||s_i||o_{i-1}||x_i) &= x_{i-1}
\end{aligned}
$$

We can extend this idea all the way back to U_0.

Constructing Astrape: Initialization

We are now ready to present an informal definition of Astrape's protocol

First, U_0 derives:

- Random strings $(r_1, \ldots, r_n)$ and $(o_1, \ldots, o_n)$
- $s_i = H(r_i \oplus r_{i_1} \oplus \cdots \oplus r_n)$
- $x_i = H(r_i||s_i||s_{i+1}||o_i||x_{i+1}); x_n = H(o_n)$

U_0 then sends to each hop U_i the tuple $(r_i, s_i, s_{i+1}, x_i, x_{i+1}, o_i)$

The last hop U_n gets (r_n, s_n, x_n, o_n).

Constructing Astrape: Sending the coins

Each U_i then sends to its successor U_{i+1} the coin $U_i \to U_{i+1}$.

This is locked with a script that allows the coin to be spent by one of two ways:

- Solving a multi-hop HTLC on r_i (the “normal” case)
- Presenting a fraud proof for any “downstream” hop, with enough information to convince anyone possessing x_i. (the “fraud” case)

Constructing Astrape: Spending the coins

After receiving a coin from U_{n-1}, U_n spends it by solving the multi-hop HTLC clause with $\pi = r_n$.

Each intermediate node U_i reacts when its right coin $U_i \to U_{i+1}$ gets spent:

- If U_{i+1} spent the HTLC case with (k_{i+1}), construct $k_i = r_i \oplus k_{i+1}$
  - If π' can spend the HTLC case for our incoming $U_{i-1} \to U_i$ coin, broadcast a transaction spending it.
  - Otherwise, U_i must have lied to us! Construct fraud proof and spend the fraud case of our incoming coin.
- Otherwise, U_{i+1} spent the fraud case with a valid fraud proof somewhere down the line.
  - We solve the fraud case of our incoming coin by copying all the parameters, except adding on all our parameters so that it can be verified by our predecessor.

This continues until all coins are spent.

Discussion and evaluation

Does Astrape accomplish our goals?

Astrape accomplishes all three goals:

- **Relationship anonymity**: harder to prove than with Fulgor/multi-hop HTLC, due to the complexity in the Collapse case. Hinges on inability to correlate x_i and x_j as long as $j > x + 1$.
- **Balance security**: $U_i \rightarrow U_{i+1}$ being spent always leads to $U_{i-1} \rightarrow U_i$ being spendable.
- **Wormhole resistance**: yes, because $U_{i-1} \rightarrow U_i$ can only be spent if $U_i \rightarrow U_{i+1}$ is spent, assuming U_{i+1} and the sender are honest.

Performance

TABLE I
RESOURCE USAGE OF DIFFERENT PCN SYSTEMS

	Plain HTLC	Fulgor/Rayo	AMHL (ECDSA)	Astrape	Astrape (BCH)
Computation time (ms)	< 0.001	$264 \cdot n$	$3 \cdot n$	$0.25 \cdot n$	$1.31 \cdot n$
Communication size (bytes)	$32 \cdot n$	$1650000 \cdot n$	$544 \cdot n$	$192 \cdot n$	$192 \cdot n$
Lock size (bytes)	$32 + c$	$32 + c$	$32 + c$	$64 + d$	$120 + 56n^2$
Unlock size, normal case (bytes)	32	32	64	32	32
Unlock size, worst case (bytes)	32	32	64	$32 \cdot n$	$32 \cdot n$

Conclusion

Astrape achieves strong anonymity and high performance while using only black-box access to a secure hash function and signature scheme.

This solves a significant open problem in the field — “what are the basic cryptographic building blocks needed for anonymous multi-hop transactions” (Malavolta 2019)

- Answer: the same building blocks needed for “usual” multi-hop transactions

We can easily deploy Astrape on legacy blockchains like Bitcoin Cash at high performance.

Questions?