A2L: Anonymous Atomic Locks for Scalability and Interoperability in Payment-Channel Hubs

Erkan Tairi¹, Pedro Moreno-Sanchez¹, Matteo Maffei¹

¹TU Vienna

Scaling Bitcoin
Tel Aviv, Sep 12th 2019
Scalability Issues

- Decentralized data structure recording each transaction in order to provide public verifiability
- Global consensus: everyone checks the whole blockchain

- Bitcoin’s transaction rate: ~10 tx/sec
- Visa’s transaction rate: ~10K tx/sec
Scalability Solutions?

- **On-chain** (tweak consensus)
  e.g., DAG Blockchain, sharding, ...

- **Off-chain** (use blockchain only for disputes)
  e.g., Payment Channel Networks

  ![Lightning Network (Bitcoin)](LND)
  ![Raiden Network (Ethereum)](RAIDEN)

  Many other projects (Bolt, Perun, Liquidity Network...)

Scalability Solutions?

- **On-chain** (tweak consensus)
  e.g., DAG Blockchain, sharding, ...

- **Off-chain** (use blockchain only for disputes)
  e.g., Payment Channel Networks

Many other projects (Bolt, Perun, Liquidity Network...)

- Lightning Network (Bitcoin)
- Raiden Network (Ethereum)
Background on Payment Channels
101 Payment Channels
A protocol to perform payments off-chain between two users
A protocol to perform payments off-chain between two users

Divided in three phases:

- **Open** channel: Deposit coins in the channel
  - Analogy: get a gift card
A protocol to perform payments off-chain between two users

Divided in three phases:

- **Open** channel: Deposit coins in the channel
  - Analogy: get a gift card
- **Pay**: Send coins off-chain by exchanging authenticated messages in a peer-to-peer fashion
  - Analogy: pay with the gift card
A protocol to perform payments off-chain between two users

Divided in three phases:

- **Open** channel: Deposit coins in the channel
  - Analogy: get a gift card
- **Pay**: Send coins off-chain by exchanging authenticated messages in a peer-to-peer fashion
  - Analogy: pay with the gift card
- **Close** channel: Redeem coins according to the last agreed state (or dispute resolution)
  - Analogy: redeem remaining coins at the gift card
Key point for scalability:
- Only open and close channel are on-chain
- Rest are off-chain operations

For more technical description:
- See talk: Pedro Moreno-Sanchez, “Atomic Multi-Channel Updates with Constant Collateral in Bitcoin-Compatible Payment-Channel Networks”, Scaling Bitcoin 2019
One cannot open channels with everyone...
⇒ exploit gateway channels!

Payment Channel Hubs (PCHs)
Payment Channel Hubs (PCHs)

One cannot open channels with everyone...
⇒ exploit gateway channels!

Alice

Send 1 BTC to Carol

Bob

...
Payment Channel Hubs (PCHs)

1. Send 1 BTC
Payment Channel Hubs (PCHs)

1. Send 1 BTC

2. Forward 1 BTC to Carol
Payment Channel Hubs (PCHs)

1. Send 1 BTC + fee to gateway

Fee acts as an incentive for gateway to participate in the payment

2. Forward 1 BTC to Carol

Gateway

Alice

Carol

Gateway

Carol
Security, Privacy and Interoperability in PCHs

A^2L: Anonymous Atomic Locks for Scalability and Interoperability in Payment Channel Hubs

Erkan Tairi, Pedro Moreno-Sanchez and Matteo Maffei
TU Wien
{erkan.tairi,pedro.sanchez,matteo.maffei}@tuwien.ac.at

Abstract—The striking growth in cryptocurrencies is revealing several scalability issues that go beyond the growing size of the blockchain. Payment channel hubs (PCHs) constitute a promising scalability solution by performing off-chain payments between sender and receiver through an intermediary, called the tumbler. While currently proposed PCHs provide security and privacy guarantees against a malicious tumbler, they fall short of other fundamental properties, such as interoperability and fungibility.

In this work, we present A^2L, the first secure, privacy-preserving, interoperable, and fungibility-preserving PCH. A^2L builds on a novel cryptographic primitive that realizes a three-party protocol for conditional transactions, where the intermediary pays the receiver only if the latter solves a cryptographic challenge with the help of the sender. We prove the security and privacy guarantees of A^2L in the Universal Composability framework and present two provably secure instantiations based on Schnorr and ECDSA signatures.

We implemented A^2L and our evaluation shows that it outperforms TumbleBit, the state-of-the-art PCH in terms of scalability. Furthermore, whenever two users open a channel, they are simply agreeing on a new distribution of the coins locked in the channel: the corresponding transactions are stored locally, that is, off-chain. When the two users disagree on the current redistribution or simply terminate their economical relation, they submit an on-chain transaction that sends back the coins to their owners according to the last agreed distribution of coins, thereby closing the channel. Thus, payment channels require only two on-chain transactions (i.e., open and close channel), yet supporting arbitrarily many off-chain payments, which significantly enhances the scalability of the underlying blockchain.

The problem with this simple construction is that in order to pay different people, a user should establish a channel with each of them, which is computationally and financially prohibitive, as this party would have to lock an amount of coins proportional to the number of users she wants to transact with.
Security in PCHs: Atomicity

1. Send 1 BTC
Security in PCHs: Atomicity

1. Alice sends 1 BTC to Gateway.

2. Gateway forwards 1 BTC to Carol.

1. Carol receives 1 BTC.

2. Gateway forwards 1 BTC to Carol.

Carol receives 1 BTC.
Security in PCHs: Atomicity

1. Send 1 BTC + fee to gateway

Gateway

Alice

3-fee

2

Gateway

1 4

2. Forward 1 BTC to Carol

Carol

Alice

3-fee

2

3-fee

2
Security in PCHs: Atomicity

1. Send 1 BTC + fee to gateway

Gateway

2. Forward 1 BTC to Carol

Gateway

Should happen atomically
Privacy in PCHs: Unlinkability
Privacy in PCHs: Unlinkability

A → E → C
A' → E → C'

pays to ≈ pays to
Interoperability in PCHs

- Create a PCH payment protocol backwards compatible with Bitcoin (and as many cryptocurrencies as possible)
- Not an easy task:
  - Perun, Liquidity Network: Ethereum-based solutions
  - BOLT: Requires opcodes not available in Bitcoin
  - TeeChain: Requires trusted execution environment (e.g., intel SGX)
  - Blind Swaps: Requires Schnorr signatures
  - TumbleBit: Compatible with Bitcoin
    - Requires HTLC: Not possible in cryptocurrencies without scripting language (e.g., Monero)
    - Efficiency can be improved
Our Approach: Building Block

- Based on adaptor signatures (AS)

Adaptor: \((pk_C, sk_C)\)

\((pk_A, sk_A)\)

\((pk_B, sk_B)\)
Our Approach: Building Block

- Based on adaptor signatures (AS)

  - Adaptor: \((pk_C, sk_C)\)
  - (pk_A, sk_A)
  - (pk_B, sk_B)

- Goals:
  - Alice can create a “half-signature” that Bob can only finish by knowing the adaptor \(sk_C\)
  - If Bob finishes the signature, Alice learns \(sk_C\)
Our Approach: Building Block

- Based on adaptor signatures (AS)

\[-\text{Adaptor: } (pk_C, sk_C)\]

\[-\text{tx}_A: \text{Alice pays 1 coin to Bob}\]

\[-(pk_A, sk_A)\]

\[-(pk_B, sk_B)\]
Our Approach: Building Block

- Based on adaptor signatures (AS)

Adaptor: $(pk_C, sk_C)$

$t_{xA}$: Alice pays 1 coin to Bob

$(pk_A, sk_A)$

$(pk_B, sk_B)$
Our Approach: Building Block

- Based on adaptor signatures (AS)

Adaptor: \((pk_C, sk_C)\)

\(tx_A: Alice\ pays\ 1\ coin\ to\ Bob\)

\((pk_A, sk_A)\)

\((pk_B, sk_B)\)

2Party AS

\(\sigma_B\)

\(sk_B\)
Our Approach: Building Block

- Based on adaptor signatures (AS)

Adaptor: \((pk_C, sk_C)\)

\(tx_A: \text{Alice pays 1 coin to Bob}\)

\((pk_A, sk_A)\) \hspace{1cm} \(sk_B\) \hspace{1cm} \(\sigma_B\)

\(\sigma, sk_C\) \hspace{1cm} \(sk_B\) \hspace{1cm} \(\sigma\)

2Party AS

\(\sigma, sk_C\)
Our Approach: Building Block

- Based on adaptor signatures (AS)

Adaptor: \((pk_C, sk_C)\)

\(tx_A:\) Alice pays 1 coin to Bob

\((pk_A, sk_A)\)

\((pk_B, sk_B)\)

2Party AS

Lock

Release
See talk: Omer Shlomovits, “Threshold Scriptless Scripts”, ScalingBitcoin 2019

See talk: Andrew Poelstra, “Workshop on Scriptless Scripts”, ScalingBitcoin 2018

More details, see paper: Malavolta et al., “Anonymous Multi-Hop Locks for Blockchain Scalability and Interoperability”, NDSS 2019. Constructions with:

- One-way homomorphic functions
- Schnorr signatures
- ECDSA (building on 2p-ECDSA of Lindell)
Our Approach: First Attempt

Adaptor: \( pk_C \)

\((pk_A, sk_A)\)

(\(pk_G, sk_G, sk_C\))

Adaptor: \( pk_C \)

\((pk_B, sk_B)\)
Our Approach: First Attempt

(pk_A, sk_A) (pk_G, sk_G, sk_C) (pk_B, sk_B)

Adaptor: pk_C

2Party AS

2Party AS
Our Approach: First Attempt

Adaptor: \( \text{pk}_C \)

\((\text{pk}_A, \text{sk}_A)\)

Adaptor: \( \text{pk}_C \)

\((\text{pk}_G, \text{sk}_G, \text{sk}_C)\)

\((\text{pk}_B, \text{sk}_B)\)

2Party AS

\(\text{tx}_G, \text{sk}_G\)

\(\sigma_B\)

\(\text{sk}_B\)
Our Approach: First Attempt

Adaptor: $pk_C$  
$sk_C$  

$tx_G, sk_G$  

$pk_A, sk_A$  

$pk_B, sk_B$  

$σ_B$  

$σ_G$  

2Party AS  

2Party AS
Our Approach: First Attempt

Adaptor: \(pk_C\)

\((pk_A, sk_A)\)  \((pk_G, sk_G, sk_C)\)  \((pk_B, sk_B)\)

2Party AS

\(tx_A, sk_A\)

\(sk_G\), \(\sigma_G\), \(sk_C\), \(\sigma\)

2Party AS

\(tx_G, sk_G\)

\(sk_B\), \(\sigma_B\)
Our Approach: First Attempt

Adaptor: \( pk_C \)

\((pk_A, sk_A)\)

\((pk_G, sk_G, sk_C)\)

\((pk_B, sk_B)\)

2Party AS

\(tx_A, sk_A\)

\(sk_G\)

\(\sigma_G\)

\(sk_C\)

\(\sigma\)

\(\sigma, sk_C\)

\(sk_B\)

\(\sigma_B\)

\(\sigma\)

\(\sigma, sk_C\)
Our Approach: First Attempt

\[(pk_A, sk_A)\] \hspace{1cm} \[(pk_G, sk_G, sk_C)\] \hspace{1cm} \[(pk_B, sk_B)\]

Adaptor: \(pk_C\)
Our Approach: First Attempt

(pk_A, sk_A) → 2Party AS → (pk_G, sk_G, sk_C) → 2Party AS → (pk_B, sk_B)

Adaptor: pk_C

σ, sk_C
Privacy in PCHs: Unlinkability

A \rightarrow pk \rightarrow E \rightarrow pk \rightarrow C

A' \rightarrow pk' \rightarrow E \rightarrow pk' \rightarrow C'

\approx

\approx

\approx

pays to

pays to

pays to

\approx

\approx

\approx

\approx
Privacy in PCHs: Unlinkability

A

\[ \text{pk} \]

A'

\[ \text{pk}' \]

C

\[ \text{pk} \]

C'

\[ \text{pk}' \]
A2L: Protocol Overview

Adaptor: \( \text{pk}_C \)

\((\text{pk}_A, \text{sk}_A, r_A)\)

\((\text{pk}_G, \text{sk}_G, \text{sk}_C)\)

\((\text{pk}_B, \text{sk}_B, r_B)\)

2Party AS

2Party AS
A2L: Protocol Overview

(pk_A, sk_A, r_A)

Adaptor: pk_C

(pk_G, sk_G, sk_C)

(pk_B, sk_B, r_B)

2Party AS

t_{x_G, sk_G Enc_G(sk_C)}

2Party AS

σ_B Enc_G(sk_C)

sk_B
A2L: Protocol Overview

Adaptor: \((pk_c)^{r_B \cdot r_A}\)

\((pk_A, sk_A, r_A)\)

\((pk_G, sk_G, sk_C)\)

Adaptor: \(pk_C\)

\((pk_B, sk_B, r_B)\)

2Party AS

\(tx_G, sk_G \ Enc_G(sk_C)\)

2Party AS

\(sk_B\)

\(\sigma_B \ Enc_G(sk_C)\)
A2L: Protocol Overview

Adaptor: \((pk_C)^{r_B \cdot r_A}\)

\((pk_A, sk_A, r_A)\)

2Party AS

\(tx_A, sk_A, Enc_G(sk_C)\)

\(Enc_G(sk_C \cdot r_B \cdot r_A)\)

\(sk_G\)

\(\sigma_G \cdot Enc_G(sk_C \cdot r_B \cdot r_A)\)

2Party AS

Adaptor: \(pk_C\)

\((pk_B, sk_B, r_B)\)

\(sk_B\)

\(\sigma_B \cdot Enc_G(sk_C)\)
A2L: Protocol Overview

Adaptor: \((pk_C) r_B \ast r_A\)

\((pk_A, sk_A, r_A)\)

2Party AS

\(tx_A, sk_A, Enc_G(sk_C)\)

\(Enc_G(sk_C \ast r_B \ast r_A)\)

\(\sigma\), \(sk_C \ast r_B \ast r_A\)

\(\sigma_G\), \(Enc_G(sk_C \ast r_B \ast r_A)\)

\(sk_G\)

\(\sigma\)

Adaptor: \(pk_C\)

\((pk_G, sk_G, sk_C)\)

2Party AS

\(tx_G, sk_G Enc_G(sk_C)\)

\(sk_B\)

\(\sigma_B\), \(Enc_G(sk_C)\)
A2L: Protocol Overview

Adaptor: \((pk_c)^{r_B} \cdot r_A\)

\((pk_A, sk_A, r_A)\)

2Party AS

\(tx_A, sk_A, Enc_G(sk_C)\)

\(sk_G\)

\(\sigma_G, Enc_G(sk_C \cdot r_B \cdot r_A)\)

\(sk_C \cdot r_B \cdot r_A\)

\(\sigma, sk_C\)

2Party AS

Adaptor: \(pk_c\)

\((pk_G, sk_G, sk_C)\)

\(tx_G, sk_G\)

\(Enc_G(sk_C)\)

\(\sigma, sk_C\)

\(\sigma\)

\(\sigma_B, Enc_G(sk_C)\)

\(sk_B\)

\(sk_C\)

\(\sigma\)
A2L: Protocol Overview

Adaptor: \((pk_C)^{r_B} * r_A\)

\((pk_A, sk_A, r_A)\)

\((pk_G, sk_G, sk_C)\)

\((pk_B, sk_B, r_B)\)

\(tx_A, sk_A, Enc_G(sk_C)\)

\(sk_G\)

\(\sigma_G\) \(Enc_G(sk_C * r_B * r_A)\)

\(sk_C * r_B * r_A\)

\(\sigma\)

\(\sigma, sk_C\)

\(\sigma\)

\(\sigma, sk_C\)
A2L: Protocol Overview

Adaptor: \( (pk_C)^{r_B} * r_A \)

\((pk_A, sk_A, r_A)\)

\((pk_G, sk_G, sk_C)\)

\((pk_B, sk_B, r_B)\)

\(tx_A, sk_A, Enc_G(sk_C)\)

\(sk_G, Enc_G(sk_C * r_B * r_A)\)

\(\sigma, sk_C * r_B * r_A\)

\(\sigma\)

\(\sigma, sk_C\)

\(sk_B, Enc_G(sk_C)\)

\(\sigma_B, Enc_G(sk_C)\)

\(sk_C\)

\(\sigma\)
Discussion

- A2L achieves atomicity and unlinkability
  - Formally proven in the UC framework
Discussion

- A2L achieves atomicity and unlinkability
  - Formally proven in the UC framework
- 2Party AS can be instantiated with One-way homomorphic functions, Schnorr or ECDSA
  - Backwards compatible with Bitcoin
  - Also compatible if Bitcoin adopts Schnorr
Discussion

- A2L achieves atomicity and unlinkability
  - Formally proven in the UC framework
- 2Party AS can be instantiated with One-way homomorphic functions, Schnorr or ECDSA
  - Backwards compatible with Bitcoin
  - Also compatible if Bitcoin adopts Schnorr
- It requires only signature verification and timelocks (instead of HTLC):
  - Interoperability with scriptless currencies (e.g., Monero)
Discussion

- A2L achieves atomicity and unlinkability
  - Formally proven in the UC framework
- 2Party AS can be instantiated with One-way homomorphic functions, Schnorr or ECDSA
  - Backwards compatible with Bitcoin
  - Also compatible if Bitcoin adopts Schnorr
- It requires only signature verification and timelocks (instead of HTLC):
  - Interoperability with scriptless currencies (e.g., Monero)
- Good for fungibility
  - Protocol results in a valid signature similar to any other transaction
  - Other information (e.g., encryptions) are not included
Evaluation

- Prototype implementation in C
- We evaluate the computation and communication overhead in LAN network
  - Comparison with TumbleBit

<table>
<thead>
<tr>
<th></th>
<th>TumbleBit</th>
<th>A2L (Schnorr)</th>
<th>A2L (ECDSA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computation Overhead</td>
<td>600 ms</td>
<td>70 ms; 8x faster</td>
<td>110 ms; 5x faster</td>
</tr>
<tr>
<td>Communication Overhead</td>
<td>326 KB</td>
<td>3.5 KB; 95x reduction</td>
<td>5 KB; 65x reduction</td>
</tr>
</tbody>
</table>

- Number of operations and communication overhead are asymptotically reduced
  - TumbleBit uses cut-and-choose
  - Size of exchanged messages grow non-linearly in the security parameter
A2L is a cryptographic protocol for PCHs that achieves security, unsinkability and interoperability.

Formally specified and proven secure in the UC Framework.

Advantages:
- Fully backwards compatible with Bitcoin (and Schnorr if adopted in Bitcoin), and scriptless cryptocurrencies (e.g., Monero)
- The most efficient Bitcoin-compatible PCH

Implementation available at https://github.com/etairi/A2L
A2L is a cryptographic protocol for PCHs that achieves security, unsinkability and interoperability

Formally specified and proven secure in the UC Framework

Advantages:
- Fully backwards compatible with Bitcoin (and Schnorr if adopted in Bitcoin), and scriptless cryptocurrencies (e.g., Monero)
- The most efficient Bitcoin-compatible PCH

Implementation available at https://github.com/etairi/A2L

THANKS! @pedrorechez