Security and Privacy for Payment Channel Networks

Pedro Moreno-Sanchez

Blockchain Summer School BDLT’19
Vienna, Sep 2nd 2019
Blockchain Research Lab: Highlights

- **CoinShuffle**: privacy-preserving protocol for blockchain payments implemented in several cryptocurrencies wallets.

- **AMHL**: first solution for security, privacy and interoperability issues with blockchain scalability protocols. Implemented in LND (current Bitcoin scalability protocol), KZen Network and COMIT Network.

- **DLSAG**: first scalability protocol with formal guarantees for the Monero cryptocurrency. Under discussion in the Monero community for adoption.

- Lots of work on:
  - Security verification and safe design of smart contracts
  - Privacy-preserving routing mechanisms
  - Constant collateral for Bitcoin-compatible PCNs
Blockchain Research Lab: Collaborations

C. Schneidewind  E. Tairi  I. Grischchenko  M. Maffei
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](LND) Lightning Network (Bitcoin)  
  ![Raiden Network](RAIDEN) Raiden Network (Ethereum)

Many other research projects (Bolt, Z-Channels, 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

  ![Lightning Network (Bitcoin)](https://example.com/lightning-network-icon)
  ![Raiden Network (Ethereum)](https://example.com/raiden-network-icon)

Many other research projects (Bolt, Z-Channels, Perun, Liquidity Network ...
Background on Payment Channel Networks
Payment Channels: Open

Blockchain

Alice

Bob
Payment Channels: Open

Blockchain

- Alice creates multisig contract to deposit money on the channel

Multisig Contract
Can be spent only with the signatures of both Alice and Bob
Alice creates multisig contract to deposit money on the channel.

Alice lets Bob sign a refund transaction to unlock the money.
Payment Channels: Open

Alice

 Blockchain

- Alice creates multisig contract to deposit money on the channel
- Alice lets Bob sign a refund transaction to unlock the money
- Alice places the multisig contract onchain
Payment Channels: Transactions

Blockchain

Alice

Bob

5 (Alice, Bob) → 4 (Alice) → 1 (Bob)

Alice ? Bob

5 (Alice)

5 (Alice, Bob) → 5 (Alice) → Alice
Payment Channels: Transactions

Blockchain

Under the hood
Mechanisms for bidirectional payments and for revocation of old states
Payment Channels: Close

Blockchain

Alice

Bob

5 (Alice)

5 (Alice, Bob)

3 (Alice)

2 (Bob)

5 (Alice, Bob)

Alice, Bob

Alice
Payment Channel Networks (PCNs)

Alice

Bob

Carol

Send 1 BTC to Carol

One cannot open channels with everyone...
⇒ exploit channel paths!
Payment Channel Networks (PCNs)

1. Send 1 BTC

Alice

Send 1 BTC to Carol

Bob

Carol

Alice

3 2

3 2

Send 1 BTC to Carol

Bob

Carol
Payment Channel Networks (PCNs)

1. Send 1 BTC to Carol

2. Forward 1 BTC to Carol
Payment Channel Networks (PCNs)

1. Send 1 BTC to Carol

2. Forward 1 BTC to Carol

Should happen atomically
Payment Channel Networks (PCNs)

1. Send 1 BTC to Carol

1. Send 1 BTC + fee to Bob

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

2. Forward 1 BTC to Carol

Should happen atomically
The Lightning Network (LN)
Hashtime Lock Contract (HTLC)
Hashtime Lock Contract (HTLC)

With knowledge of $x$, Bob can "open" + publish the transaction on the blockchain for enforcing the payment.
Hashtime Lock Contract (HTLC)

With knowledge of $x$, Bob can “open” + publish the transaction on the blockchain for enforcing the payment.

After time the transaction cannot be published anymore on the blockchain.

Diagram:
- Alice
- Bob
- Transaction process:
  1. Alice and Bob
  2. After time
  3. Alice and Bob
  4. Alice
  5. Alice and Bob
Hashtime Lock Contract (HTLC)

HTLC (Alice, Bob, 1, y, t):
Alice pays Bob 1 BTC iff Bob shows some x such that H(x) = y before t.

With knowledge of x, Bob can “open” + publish the transaction on the blockchain for enforcing the payment.

After time the transaction cannot be published anymore on the blockchain.
HTLC for Multi-hop Payments

Alice

2 3

Bob

3 2

Carol

y := H(x)
HTLC for Multi-hop Payments

\[ y := H(x) \]
HTLC for Multi-hop Payments

Alice

Bob

Carol

HTLC(Alice, Bob, 1.1, y, t)

y

y := H(x)
HTLC for Multi-hop Payments

$$y := H(x)$$

HTLC(Alice, Bob, 1.1, y, t)

HTLC(Bob, Carol, 1, y, t')
HTLC for Multi-hop Payments

HTLC(Alice, Bob, 1.1, y, t)  
HTLC(Bob, Carol, 1, y, t')  

y := H(x)
HTLC for Multi-hop Payments

HTLC(Alice, Bob, 1.1, y, t)

HTLC(Bob, Carol, 1, y, t')

y := H(x)
HTLC for Multi-hop Payments

**Requirement:** \( t > t' \)

(after Carol revealed \( x \) to Bob, there must still be time for Bob to reveal \( x \) to Alice)

\[
\text{HTLC(Alice, Bob, 1.1, y, t)} \quad \text{HTLC(Bob, Carol, 1, y, t')} \n\]

\[
y := H(x) \]

\[
\text{Alice} \quad 0.9 \quad 4.1 \quad \text{Bob} \quad 2 \quad 3 \quad \text{Carol} \]

\[
\text{Alice} \xrightarrow{x} \text{Bob} \xrightarrow{x} \text{Carol} \]

\[
\text{y} := H(x) \]
Lightning Network & Co work allow us to perform payments offchain

- fast, no confirmation delay
- little fees
- minimal information stored on the blockchain
- secure and privacy-preserving (at a first glance...)

The blockchain is used only to mediate disputes...cool!

**HTLC (Alice, Bob, 1.1, y, t):**
Alice pays Bob 1.1 BTC iff Bob shows some x such that H(x) = y before t days
Security + Privacy in PCNs

Are off-chain payments in PCNs secure?
(No honest participant loses money)

Are off-chain payments in PCNs privacy-preserving by default?
(individual payments are not recorded on the blockchain)
Security + Privacy in PCNs

Are off-chain payments in PCNs secure? (No honest participant looses money)

NO!

Are off-chain payments in PCNs privacy-preserving by default? (individual payments are not recorded on the blockchain)

NO!
Security and Privacy Issues in Existing PCNs

Concurrency and Privacy with Payment-Channel Networks*

Giulio Malavolta1
Friedrich-Alexander-University Erlangen-Nürnberg
malavolta@cs.lau.de

Pedro Moreno-Sanchez1
Purdue University
pmoreno@purdue.edu

Aniket Kate
Purdue University
anket@purdue.edu

Matteo Maffei
TU Wien
matteo.maffei@tuwien.ac.at

Srivas Ravi
University of Southern California
srivarat@usc.edu

Abstract

Permissionless blockchains protocols such as Bitcoin are inherently limited in transaction throughput and latency. Current efforts to address this key issue focus on off-chain payment channels that can be combined in a Payment-Channel Network (PCN) to enable an unlimited number of payments without requiring to access the blockchain other than to register the initial and final capacity of each channel. While this approach paves the way for low latency and high throughput of payments, its deployment in practice raises several privacy concerns as well as technical challenges related to the inherently concurrent nature of payments that have not been sufficiently studied as far.

In this work, we lay the foundations for privacy and concurrency in PCNs, presenting a formal definition in the Universal Composability framework as well as practical and provably secure solutions. In particular, we present Fulgor and Rayo. Fulgor is the first payment protocol for PCNs that provides provable privacy guarantees for PCNs and is fully compatible with the Bitcoin scripting system. However, Fulgor is a blocking protocol and therefore prone to deadlocks of concurrent payments as in currently available PCNs. Instead, Rayo is the first protocol for PCNs that enforces non-blocking progress (i.e., at least one of the concurrent payments terminates). We show through a new impossibility result that non-blocking

Anonymous Multi-Hop Locks for Blockchain Scalability and Interoperability

Giulio Malavolta1,2, Pedro Moreno-Sanchez1,2,3, Clara Schneidewind1, Aniket Kate1, Matteo Maffei1

Friedrich-Alexander-University Erlangen-Nürnberg, 1TU Wien, 2Purdue University

Abstract—Tremendous growth in cryptocurrency usage is exposing the inherent scalability issues with permissionless blockchain technology. Payment-channel networks (PCNs) have emerged as the most widely deployed solution to mitigate the scalability issues, allowing the bulk of payments between two users to be carried out off-chain. Unfortunately, as reported in the literature and further demonstrated in this paper, current PCNs do not provide meaningful security and privacy guarantees [32], [42].

In this work, we study and design secure and privacy-preserving PCNs. We start with a security analysis of existing PCNs, reporting a new attack that applies to all major PCNs, including the Lightning Network, and allows an attacker to steal the fees from honest intermediaries in the same payment path. We then formally define anonymous multi-hop locks (AMHLs), a novel cryptographic primitive that serves as a cornerstone for the design of secure and privacy-preserving PCNs. We present several provably secure cryptographic instantiations that make AMHLs compatible with the vast majority of cryptocurrencies. In particular, we show that (linear) homomorphic one-way functions suffice to construct AMHLs for PCNs supporting

I. INTRODUCTION

Cryptocurrencies are growing in popularity and are playing an increasing role in the worldwide financial ecosystem. In fact, the number of Bitcoin transactions grew by approximately 30% in 2017, reaching a peak of more than 420,000 transactions per day in December 2017 [2]. This striking increase in demand has given rise to scalability issues [20], which go well beyond the rapidly increasing size of the blockchain. For instance, the permissionless nature of the consensus algorithm used in Bitcoin today limits the transaction rate to tens of transactions per second, whereas other payment networks such as Visa support peaks of up to 47,000 transactions per second [9].

Among the various proposals to solve the scalability issue [22], [23], [40], [50], payment-channels have emerged as the most widely deployed solution in practice. In a nutshell, two users open a payment channel by committing a single transaction to the blockchain, which locks their bitcoins in a deposit secured by a
Security Issue: The Wormhole Attack

\[ y := H(x) \]
Security Issue: The Wormhole Attack

\[ \text{HTLC}(A, E_1, 1.3, y, t_1) \]
\[ \text{HTLC}(E_1, B, 1.2, y, t_2) \]
\[ \text{HTLC}(B, E_2, 1.1, y, t_3) \]
\[ \text{HTLC}(E_2, C, 1, y, t_4) \]

\[ y := H(x) \]
Security Issue: The Wormhole Attack

$\text{HTLC}(A, E_1, 1.3, y_1, t_1)$

$\text{HTLC}(E_1, B, 1.2, y_2, t_2)$

$\text{HTLC}(B, E_2, 1.1, y_3, t_3)$

$\text{HTLC}(E_2, C, 1, y_4, t_4)$

$y := H(x)$
Security Issue: The Wormhole Attack

\[ \text{HTLC}(A, E_1, 1.3, y, t_1) \]
\[ \text{HTLC}(E_1, B, 1.2, y, t_2) \]
\[ \text{HTLC}(B, E_2, 1.1, y, t_3) \]
\[ \text{HTLC}(E_2, C, 1, y, t_4) \]

\[ y := H(x) \]
Security Issue: The Wormhole Attack

B considers the payment to be failed and unlocks his funds after the timeout

HTLC(A, E₁, 1.3, y, t₁)
HTLC(E₁, B, 1.2, y, t₂)
HTLC(B, E₂, 1.1, y, t₃)
HTLC(E₂, C, 1, y, t₄)

y := H(x)
Security Issue: The Wormhole Attack

HTLC(A, E₁, 1.3, y, t₁)
HTLC(E₁, B, 1.2, y, t₂)
HTLC(B, E₂, 1.1, y, t₃)
HTLC(E₂, C, 1, y, t₄)

B considers the payment to be failed and unlocks his funds after the timeout

A \xrightarrow{x} \quad E₁ \quad \xleftarrow{x} \quad B \quad \xrightarrow{x} \quad E₂ \quad \xrightarrow{x} \quad C

gets 1.3 (no payment to B)

pays 1 (no payment from B)

\text{Attacker earns 0.3 BTC (own fees + B's fee)}

y \:= H(x)
Privacy Issues in HTLC Payments

**Relationship Anonymity:** On-path adversaries do not learn who pays to whom
Privacy Issues in HTLC Payments

Relationship Anonymity: On-path adversaries do not learn who pays to whom
Privacy Issues in HTLC Payments

HTLC(A, E₁, v₁, y₁, t₁)

HTLC(E₁, B, v₂, y₂, t₂)

HTLC(B, E₂, v₃, y₃, t₃)

HTLC(E₂, C, v₄, y₄, t₄)

A

E₁

B

E₂

C

A’

C’

HTLC(A, E₁, v₁, y₂, t₁)

HTLC(E₁, B, v₂, y₂, t₂)

HTLC(B, E₂, v₃, y₃, t₃)

HTLC(E₂, C, v₄, y₄, t₄)

Relationship Anonymity: On-path adversaries do not learn who pays to whom
Privacy Issues in HTLC Payments

Relationship Anonymity: On-path adversaries do not learn who pays to whom
Solving Security and Privacy Issues in Payment Channel Networks
Solving Security + Privacy Issues

Randomised conditions at each hop that can only be released by (exactly) the right neighbour’s key.
Solving Security + Privacy Issues

Setup phase for the distribution of individual “randomisation factors” for users at each hop

Randomised conditions at each hop that can only be released by (exactly) the right neighbour’s key

\[
\text{Lock}(A, E_1, 1.3, C_1, t_1) \\
\text{Lock}(E_1, B, 1.2, C_2, t_2) \\
\text{Lock}(B, E_2, 1.1, C_3, t_3) \\
\text{Lock}(E_2, C, 1, C_4, t_4)
\]
Solving Security + Privacy Issues

Setup phase for the distribution of individual “randomisation factors” for users at each hop

Randomised conditions at each hop that can only be released by (exactly) the right neighbour’s key

1. **Atomicity:**
   - If a user’s right lock gets opened, he can open his left lock

2. **Consistency:**
   - A user can open his left lock only if his right lock was released

3. **Relationship Anonymity:**
   - A user learns about no other participant of the payment path than his direct neighbours

Desired Properties

- No coin loss
- No Wormhole Attacks
- Privacy
Anonymous Multi-hop-Locks (AMHL)

Ideal functionality
(capturing atomicity, consistency + relationship anonymity)

provably realise in the UC framework

Construction from homographic one-way functions

Schnorr-based construction

ECDSA-based construction
Anonymous Multi-hop-Locks (AMHL)

Ideal functionality (capturing atomicity, consistency + relationship anonymity)

provably realise in the UC framework

Construction from homographic one-way functions

Schnorr-based construction

ECDSA-based construction compatible with Bitcoin, Ethereum, etc.
ECDSA-based Secure PCNs
Scriptless Scripts
Scriptless Scripts

Alice (sk_A)

Bob (sk_B)

hypothesical “shared identity”

\[ sk_{AB} = sk_A \times sk_B \]
Scriptless Scripts

Blockchain

hypothesised “shared identity”

$$sk_{AB} = sk_A \times sk_B$$
Scriptless Scripts

Alice can retrieve secret $k$ from full signature

Bob gets sufficient information for checking that the “half signature” produced by Alice and Bob can be completed to a valid signature given $k$

$hypothetical \ "shared identity\"

$sk_{AB} = sk_A \times sk_B$

Alice (sk$_A$)

Bob (sk$_B$)

Blockchain
Extension to Multi-hop Locks

```
Lock(A, E1, 1.3, C1, t1)
(k1, C1)
Lock(E1, B, 1.2, C2, t2)
(k2, C2)
Lock(B, E2, 1.1, C3, t3)
(k3, C3)
Lock(E2, C, 1, C4, t4)
(k4, C4)
(k1 + k2 + k3 + k4)

A

E1

B

E2

C
```
Extension to Multi-hop Locks

\[
\begin{align*}
\text{Lock}(A, E_1, 1.3, C_1, t_1) & \quad \Rightarrow (k_1, C_1) \\
\text{Lock}(E_1, B, 1.2, C_2, t_2) & \quad \Rightarrow (k_1 + k_2, C_2) \\
\text{Lock}(B, E_2, 1.1, C_3, t_3) & \quad \Rightarrow (k_1 + k_2 + k_3, C_3) \\
\text{Lock}(E_2, C, 1, C_4, t_4) & \quad \Rightarrow (k_1 + k_2 + k_3 + k_4, C_4) 
\end{align*}
\]
Extension to Multi-hop Locks

\[
\text{Lock}(A, E_1, 1.3, C_1, t_1) \Rightarrow \text{Lock}(E_1, B, 1.2, C_2, t_2) \Rightarrow \text{Lock}(B, E_2, 1.1, C_3, t_3) \Rightarrow \text{Lock}(E_2, C, 1, C_4, t_4)
\]

\[
(k_2, C_2) \quad (k_3, C_3) \quad (k_4, C_4) \quad (k_1 + k_2 + k_3 + k_4)
\]

\[
k_1 \cdot G \\
(k_1 + k_2) \cdot G \\
(k_1 + k_2 + k_3) \cdot G \\
(k_1 + k_2 + k_3 + k_4) \cdot G
\]
Extension to Multi-hop Locks

\[
\begin{align*}
&\text{Lock}(A, E_1, 1.3, C_1, t_1) \\
&\text{Lock}(E_1, B, 1.2, C_2, t_2) \\
&\text{Lock}(B, E_2, 1.1, C_3, t_3) \\
&\text{Lock}(E_2, C, 1, C_4, t_4)
\end{align*}
\]
Extension to Multi-hop Locks

- Lock(A, E₁, 1.3, C₁, t₁)
  - k₁*G
- Lock(E₁, B, 1.2, C₂, t₂)
  - (k₁ + k₂)*G
- Lock(B, E₂, 1.1, C₃, t₃)
  - (k₁ + k₂ + k₃)*G
- Lock(E₂, C, 1, C₄, t₄)
  - (k₁ + k₂ + k₃ + k₄)*G

A → E₁ → B → E₂ → C

- k₁
- (k₁ + k₂)
- (k₁ + k₂ + k₃)
- (k₁ + k₂ + k₃ + k₄)

- k₂
- - k₃
- - k₄
Extension to Multi-hop Locks

Conditions look random (as they differ by a secret random factor)

A valid key can only be extracted from a valid key for the right lock
ECDSA-based Scriptless Lock

Signature w.r.t. a (public) random elliptic curve point $R$

$\sigma_R = \text{sign}(r, sk, \text{transaction})$

- secret randomness
- secret key
- message

$R = r \times G$
ECDSA-based Scriptless Lock

Signature w.r.t. a (public) random elliptic curve point $R$

$\sigma_R = \text{sign}(r, sk, \text{transaction})$

shared signature using a shared key and a shared randomness

$R = r \cdot G$

$\sqrt{AB}$

$r_A \cdot r_B \cdot G$

$r_A \cdot r_B$

$sk_A \cdot sk_B$
ECDSA-based Scriptless Lock

Signature w.r.t. a (public) random elliptic curve point $R$

$\sigma_R = \text{sign}(r, sk, \text{transaction})$

shared signature using a shared key and a shared randomness

embedding of random share (condition) $k$
ECDSA-based Scriptless Lock

Signature w.r.t. a (public) random elliptic curve point $R$

$\sigma_R = \text{sign}(r, sk, \text{transaction})$

shared signature using a shared key and a shared randomness

embedding of random share (condition) $k$

“half signature” without $k$ but still with respect to $r_A r_B k G$
ECDSA-based Scriptless Lock

Signature w.r.t. a (public) random elliptic curve point R

\[ \sigma_R = \text{sign}(r, sk, \text{transaction}) \]

shared signature using a shared key and a shared randomness

embedding of random share (condition) \( k \)

“half signature” without \( k \) but still with respect to \( r_A r_B k G \)

Lock Protocol

\( (sk_A, r_A) \) \( C = k*G, \text{transaction} \) \( (sk_B, r_B) \)

“1/3” signature \( \sigma_{R,B} \)

“1/3” signature \( \sigma_{R,A} \)
ECDSA-based Scriptless Lock

**Signature w.r.t. a (public) random elliptic curve point R**

\[ \sigma_R = \text{sign}(r, sk, \text{transaction}) \]

**Lock Protocol**

- **(sk_A, r_A)**
- **C = k*G, transaction**
- Hard for ECDSA as \( \sigma_R \) has a non-linear structure
- **(sk_B, r_B)**
- **Security** and **Privacy** proven formally (in the UC Framework)
- Compatible with Bitcoin and current PCNs
  - ✓ Implemented in
    - ✓ Lightning Network ([https://github.com/cfromknecht/tpec](https://github.com/cfromknecht/tpec))
    - ✓ COMIT Network ([https://github.com/coblox/ss-ecdsa-poc](https://github.com/coblox/ss-ecdsa-poc))
- Reduces transaction size for conditional payments
  - ✓ Encoding of condition within signature
- Makes settlement transactions indistinguishable from regular ones ([Fungibility](#))
- Little overhead:
  - ✓ < 500 bytes communication
  - ✓ few ms computation
AMHLs are suitable for cross-currency usage - even with different primitive instantiations

- Inter-currency payment channels
- Atomic swaps
Summary

The **Wormhole Attack**: A novel attack on Payment Channel Network Security

**AMHLs**: A *new primitive* for secure + anonymous Payment Channel Networks

Concrete *constructions* of AMHLs that

- ... are efficient
- ... got implemented in Bitcoin’s Lightning Network
- ... enable inter-blockchain Payment Channels