Universal Atomic Swaps: Fair Exchange of Coins Across All Blockchains

S.A. Thyagarajan, G. Malavolta, P. Moreno-Sanchez

Crosschain Workshop 2021
March 10th, 2021
- There exist many cryptocurrencies today

- One crucial use case is the exchange of coins (atomic swaps)

- Challenge: Each cryptocurrency provides different functionality in their scripting language
State-of-the-art: Atomic swap based on HTLC contract

**HTLC (Alice, Bob, 1, y, ℹ️):**
Alice pays Bob 1 BTC iff Bob shows some \( x \) such that \( H(x) = y \) before 🧵.
State-of-the-art: Atomic swap based on HTLC contract

\[ y := H(x) \]

HTLC (Alice, Bob, 1, y, ᵇ): Alice pays Bob 1 BTC iff Bob shows some x such that H(x) = y before 🤡.
State-of-the-art: Atomic swap based on HTLC contract

$y := H(x)$

$c_1 := \text{HTLC}(A, B, 1\text{BTC}, y, t)$

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

\[ y := H(x) \]

\[ c_1 := \text{HTLC}(A, B, 1\text{BTC}, y, t) \]

\[ c_2 := \text{HTLC}(B, A, 1\text{ETH}, y, t') \]
Alice pays Bob 1 BTC iff Bob shows some $x$ such that $H(x) = y$ before

Fair Exchange. Two possible outcomes:
- If Alice claims $c_2$, she shows $x$ and Bob claims $c_1$
- If Bob refunds $c_2$, Alice refunds $c_1$
HTLC Drawbacks
HTLC Drawbacks

- Compatibility of the hash function
  - Both ledgers must support the same hash function (e.g., SHA-256)
  - Both ledgers must use the same number of bits to represent x
  - Using the same value h at two ledgers is a privacy leakage!
HTLC Drawbacks

- Compatibility of the hash function
  - Both ledgers must support the same hash function (e.g., SHA-256)
  - Both ledgers must use the same number of bits to represent \( x \)
  - Using the same value \( h \) at two ledgers is a privacy leakage!
HTLC Drawbacks

- Compatibility of the hash function
  - Both ledgers must support the same hash function (e.g., SHA-256)
  - Both ledgers must use the same number of bits to represent $x$
  - Using the same value $h$ at two ledgers is a privacy leakage!

- Compatibility of time functions
  - Both ledgers must support timelock mechanism in their scripting language
  - Hinders fungibility: transactions with timelock easily distinguishable
HTLC Drawbacks

- Compatibility of the hash function
  - Both ledgers must support the same hash function (e.g., SHA-256)
  - Both ledgers must use the same number of bits to represent $x$
  - Using the same value $h$ at two ledgers is a privacy leakage!

- Compatibility of time functions
  - Both ledgers must support timelock mechanism in their scripting language
  - Hinders fungibility: transactions with timelock easily distinguishable
HTLC Drawbacks

- Compatibility of the hash function
  - Both ledgers must support the same hash function (e.g., SHA-256)
  - Both ledgers must use the same number of bits to represent $x$
  - Using the same value $h$ at two ledgers is a privacy leakage!

- Compatibility of time functions
  - Both ledgers must support timelock mechanism in their scripting language
  - Hinders fungibility: transactions with timelock easily distinguishable

- Lack of multi-asset support
  - So far, only one asset at a time
  - Hinders scenarios swapping multiple assets
HTLC Drawbacks

- Compatibility of the hash function
  - Both ledgers must support the same hash function (e.g., SHA-256)
  - Both ledgers must use the same number of bits to represent $x$
  - Using the same value $h$ at two ledgers is a privacy leakage!

- Compatibility of time functions
  - Both ledgers must support timelock mechanism in their scripting language
  - Hinders fungibility: transactions with timelock easily distinguishable

- Lack of multi-asset support
  - So far, only one asset at a time
  - Hinders scenarios swapping multiple assets

Our Approach
HTLC Drawbacks

- Compatibility of the hash function
  - Both ledgers must support the same hash function (e.g., SHA-256)
  - Both ledgers must use the same number of bits to represent $x$
  - Using the same value $h$ at two ledgers is a privacy leakage!

- Compatibility of time functions
  - Both ledgers must support timelock mechanism in their scripting language
  - Hinders fungibility: transactions with timelock easily distinguishable

- Lack of multi-asset support
  - So far, only one asset at a time
  - Hinders scenarios swapping multiple assets

Our Approach

Only digital signature support
HTLC Drawbacks

- Compatibility of the hash function
  - Both ledgers must support the same hash function (e.g., SHA-256)
  - Both ledgers must use the same number of bits to represent $x$
  - Using the same value $h$ at two ledgers is a privacy leakage!

- Compatibility of time functions
  - Both ledgers must support timelock mechanism in their scripting language
  - Hinders fungibility: transactions with timelock easily distinguishable

- Lack of multi-asset support
  - So far, only one asset at a time
  - Hinders scenarios swapping multiple assets

Our Approach

- Only digital signature support
- Timelock handled outside script
- **Compatibility of the hash function**
  - Both ledgers must support the same hash function (e.g., SHA-256)
  - Both ledgers must use the same number of bits to represent $x$
  - Using the same value $h$ at two ledgers is a privacy leakage!

- **Compatibility of time functions**
  - Both ledgers must support timelock mechanism in their scripting language
  - Hinders fungibility: transactions with timelock easily distinguishable

- **Lack of multi-asset support**
  - So far, only one asset at a time
  - Hinders scenarios swapping multiple assets

---

**Our Approach**

- Only digital signature support
- Timelock handled outside script
- Support for n-m coin swaps
Adaptor signatures instead of hash functions

CreateAS (tx, Y) \rightarrow “half-signature”

Half-signature is an almost valid signature that becomes valid if a value $y$ is attached to the signature such that $Y = y \cdot G$
Our Approach: Building Blocks

**Adaptor signatures instead of hash functions**

CreateAS (tx, Y) —> “half-signature”

- Half-signature is an almost valid signature that becomes valid if a value $y$ is attached to the signature such that $Y = y \cdot G$

$Y := y \cdot G$

Alice

Bob
Our Approach: Building Blocks

Adaptor signatures instead of hash functions

CreateAS (tx, Y) → “half-signature”
Half-signature is an almost valid signature that becomes valid if a value y is attached to the signature such that Y=y*G

Alice

Y:= y*G

CreateAS (tx, Y)

Bob
Adaptor signatures instead of hash functions

\[ \text{CreateAS (tx, Y)} \rightarrow \text{“half-signature”} \]

Half-signature is an almost valid signature that becomes valid if a value \( y \) is attached to the signature such that \( Y = y \cdot G \)

\[ Y := y \cdot G \]
Adaptor signatures instead of hash functions

CreateAS (tx, Y) \rightarrow \text{“half-signature”}

Half-signature is an almost valid signature that becomes valid if a value \( y \) is attached to the signature such that \( Y = y \cdot G \)

- Efficient constructions in ECDSA, Schnorr and linkable ring signatures (as in Monero)
- Possible to build from any digital signature (e.g., BLS) with a less efficient construction
Adaptor signatures instead of hash functions

CreateAS \((tx, Y) \rightarrow \text{“half-signature”}\)

- Half-signature is an almost valid signature that becomes valid if a value \(y\) is attached to the signature such that \(Y=y\cdot G\)

- Efficient constructions in ECDSA, Schnorr and linkable ring signatures (as in Monero)
- Possible to build from any digital signature (e.g., BLS) with a less efficient construction

Timelock puzzles instead of script-based timelocks

CreatePuzzle \((\text{refund}, T) \rightarrow \text{puzzle}\)

- Given a puzzle, one cannot extract the value inside (refund) before \(T\) sequential operations are performed
Our Approach: Building Blocks

**Adaptor signatures instead of hash functions**

CreateAS \((tx, Y)\) —> “half-signature”

- Half-signature is an almost valid signature that becomes valid if a value \(y\) is attached to the signature such that \(Y=y*G\)

- Efficient constructions in ECDSA, Schnorr and linkable ring signatures (as in Monero)
- Possible to build from any digital signature (e.g., BLS) with a less efficient construction

**Timelock puzzles instead of script-based timelocks**

CreatePuzzle \((refund, T)\) —> puzzle

- Given a puzzle, one cannot extract the value inside \((refund)\) before \(T\) sequential operations are performed

CreateAS \((tx, Y)\)

CreatePuzzle \((tx'\text{Refund}, T)\)

Y := y*G

Alice

Bob

CreatePuzzle \((tx\text{Refund}, T)\)
**Adaptor signatures instead of hash functions**

CreateAS \( (tx, Y) \rightarrow \text{“half-signature”} \)

- 
  - Efficient constructions in ECDSA, Schnorr and linkable ring signatures (as in Monero)
  - Possible to build from any digital signature (e.g., BLS) with a less efficient construction

**Timelock puzzles instead of script-based timelocks**

CreatePuzzle \( (\text{refund}, T) \rightarrow \text{puzzle} \)

Given a puzzle, one cannot extract the value inside (refund) before \( T \) sequential operations are performed
Our Approach: Building Blocks

Adaptor signatures instead of hash functions

CreateAS (tx, Y) $\rightarrow$ “half-signature”

- Half-signature is an almost valid signature that becomes valid if a value $y$ is attached to the signature such that $Y=y*G$

- Efficient constructions in ECDSA, Schnorr and linkable ring signatures (as in Monero)

- Possible to build from any digital signature (e.g., BLS) with a less efficient construction

Timelock puzzles instead of script-based timelocks

CreatePuzzle (refund, T) $\rightarrow$ puzzle

- Given a puzzle, one cannot extract the value inside (refund) before $T$ sequential operations are performed

Support of multi-asset swaps

- Replicate the puzzle for each swap

- Replicate the AS for each swap
Our Approach: Building Blocks

**Adaptor signatures instead of hash functions**

CreateAS (tx, Y) → “half-signature”

- Half-signature is an almost valid signature that becomes valid if a value y is attached to the signature such that Y=y*G
- Efficient constructions in ECDSA, Schnorr and linkable ring signatures (as in Monero)
- Possible to build from any digital signature (e.g., BLS) with a less efficient construction

**Timelock puzzles instead of script-based timelocks**

CreatePuzzle (refund, T) → puzzle

- Given a puzzle, one cannot extract the value inside (refund) before T sequential operations are performed
- Support of multi-asset swaps
  - Replicate the puzzle for each swap
  - Replicate the AS for each swap
Our Approach: Building Blocks

**Adaptor signatures instead of hash functions**

CreateAS (tx, Y) -> “half-signature”

- Efficient constructions in ECDSA, Schnorr and linkable ring signatures (as in Monero)
- Possible to build from any digital signature (e.g., BLS) with a less efficient construction

**Timelock puzzles instead of script-based timelocks**

CreatePuzzle (refund, T) -> puzzle

- Given a puzzle, one cannot extract the value inside (refund) before T sequential operations are performed

Support of multi-asset swaps

- Replicate the puzzle for each swap
- Replicate the AS for each swap
Our Approach: Building Blocks

Adaptor signatures instead of hash functions

CreateAS (tx, Y) → “half-signature”

- Efficient constructions in ECDSA, Schnorr and linkable ring signatures (as in Monero)
- Possible to build from any digital signature (e.g., BLS) with a less efficient construction

Y := y*G

CreatePuzzle(tx’Refund, T)

CreatePuzzle(txRefund, T)

CreatePuzzle(tx”Refund, T)

Puzzles

Alice

Bob

Support of multi-asset swaps

- Replicate the puzzle for each swap
- Replicate the AS for each swap

Timelock puzzles instead of script-based timelocks

CreatePuzzle (refund, T) → puzzle

Given a puzzle, one cannot extract the value inside (refund) before T sequential operations are performed
Conclusions

- There exists many cryptocurrencies today and exchange of coins (atomic swaps) is a crucial use case

- HTLC-based atomic swaps is state-of-the-art. Drawbacks
  - Compatibility with hash function
  - Support for time locks in scripting language
  - Facilitates 1-1 atomic swap

- In this work, we provide universal atomic swaps:
  - Requiring support for digital signatures only from the scripting language
  - Improves the (on-chain) privacy of atomic swaps
  - Supports for the first time n-m swaps natively in the corresponding cryptocurrencies
Pedro Moreno-Sanchez
pedro.morenosanchez@imdea.org
@pedrorechez
software.imdea.org