Keychat: Secure Messaging via Bitcoin

Robert Chen, John Kuszmaul, Yiming Zheng
Mentored By Alin Tomescu
Motivation

- We want to secure communications.
End-to-end Encryption

- People generate a key pair. They broadcast public keys.

Alice

$C = PK_B(Financial\ Information)$

Bob

$SK_B(C) = Financial\ Information$

I can’t decrypt this because I don’t have Bob’s secret key.
The Problem: Public Key Distribution

- People can encrypt messages, but they might be encrypting them for the wrong person.
The Current State of Common Communications Apps

- Facebook Messenger, Gmail - Do not use end-to-end encryption by default
- WhatsApp, Signal - Public keys are stored in a central directory that may be insecure.

Alice: What is Bob’s Public Key?

Signal: PKD

I know! It’s $PK_B$. 
Are append-only PKDs enough for security?

If append-only, then Bob can detect fake $PK_M$

*What else can the malicious PKD do?*

<table>
<thead>
<tr>
<th>Public Key Directory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>Alice</td>
</tr>
<tr>
<td>Bob</td>
</tr>
</tbody>
</table>
Our Work: Public Key Directory Equivocation

<table>
<thead>
<tr>
<th>Name</th>
<th>Public Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>( PK_A )</td>
</tr>
</tbody>
</table>

Alice’s Perspective

<table>
<thead>
<tr>
<th>Name</th>
<th>Public Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>( PK_A )</td>
</tr>
<tr>
<td>Bob</td>
<td>( PK_M )</td>
</tr>
</tbody>
</table>

Bob’s Perspective

<table>
<thead>
<tr>
<th>Name</th>
<th>Public Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>( PK_A )</td>
</tr>
<tr>
<td>Bob</td>
<td>( PK_B )</td>
</tr>
</tbody>
</table>
Outline

Keybase

Bitcoin

Catena

Keychat

Keybase

Optimizes

Witnesses

Catena

Bitcoin
Keybase: A public key directory

Keybase PKD Server

\[ S_1 = H(DIR_1) \]

<table>
<thead>
<tr>
<th>Name</th>
<th>Public Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>PK_J</td>
</tr>
<tr>
<td>Robert</td>
<td>PK_R</td>
</tr>
</tbody>
</table>

\[ S_2 = H(DIR_2) \]

<table>
<thead>
<tr>
<th>Name</th>
<th>Public Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>PK_J</td>
</tr>
<tr>
<td>Robert</td>
<td>PK_R</td>
</tr>
<tr>
<td>Yiming</td>
<td>PK_Y</td>
</tr>
</tbody>
</table>
Keybase Summaries

Yiming $S_1$

(1) Yiming PK?
(2) $PK_Y$
(3) Verify $PK_Y$ against $S_1$

Robert $S_1$

(1) Yiming PK?
(2) $PK_Y$
(3) Verify $PK_Y$ against $S_1$
Keybase Equivocation

Yiming $S_1, S_2$

1. Yiming PK?

2. $PK_Y$

3. Verify $PK_Y$ against $S_2$

Robert $S_1, S_2'$

1. Yiming PK?

2. $PK_M$

3. Verify $PK_M$ against $S_2'$
Keybase Non-equivocation

- Important that Yiming and Robert have the same history of hashes/summaries: $S_1, S_2, S_3 \ldots$

Keybase PKD Server

S_1, S_2, S_3 \ldots

Bitcoin

Download $S_1, S_2, S_3 \ldots$

Robert's, Yiming's Keychat apps
Outline

Keybase

Bitcoin

Catena

Keychat

Keybase Optimizes Witnesses Catena

Witnesses

Bitcoin
Bitcoin: Blockchain

Block 1

A → B, 5 BTC
C → D, 10 BTC
Bitcoin: Blockchain

Block 1

A → B, 5 BTC
C → D, 10 BTC

Block 2

D → B, 2 BTC
A → B, 1 BTC
Bitcoin: Blockchain

Block 1
- A → B, 5 BTC
- C → D, 10 BTC

Block 2
- D → B, 2 BTC
- A → B, 1 BTC

Block n
- M → A, 2 BTC
However, Can Forks Happen?

Block 1

A → B, 5 BTC
C → D, 10 BTC

Block 2

D → B, 2 BTC
A → B, 1 BTC

Block n

M → A, 2 BTC

Block n’

M → B, 2 BTC
However, Can Forks Happen?  

Answer: No!

Block 1

A → B, 5 BTC
C → D, 10 BTC

Block 2

D → B, 2 BTC
A → B, 1 BTC

Block n

M → A, 2 BTC

Block n'

M → B, 2 BTC
Keybase “Witnessing” Summaries

Block $i$
- $A \rightarrow B$, 5 BTC
- $C \rightarrow D$, 10 BTC

Block $j$
- $D \rightarrow B$, 2 BTC
- $A \rightarrow B$, 1 BTC

Block $n$
- $M \rightarrow A$, 2 BTC

$K \rightarrow K$, 1 BTC

$S_1$

$K \rightarrow K$, 1 BTC

$S_2$

$K \rightarrow K$, 1 BTC

$S_3$
Equivocation Within a Block?

Block j

D → B, 2 BTC
A → B, 1 BTC

K → K, 1 BTC

S_2

K → K, 1 BTC

S_2'

20
Hashes: A Reminder

SHA 256
(Hash Function)
Hashes: A Reminder

654324748 → SHA 256 (Hash Function)
Hashes: A Reminder

654324748 \rightarrow \text{SHA 256 (Hash Function)} \rightarrow \text{'46f4e4edb6612295a20af8d927b6416a59398091dd82601668e441ac0bf26e2e'}
More Properties of Ideal Hash Functions

SHA 256 (Hash Function)

654324748

SHA 256 (Hash Function)

'46f4e4ed612295a20af8d927b6416a59398091dd82601668e441ac0bf26e2e'

Always 256 bits

Bitcoin block

C → D, 10 BTC

SHA 256 (Hash Function)

'b7a4c0d08a9f74f47ce44d1f7a58d9344b35aab00c19975828ff6b5e556e6fa'

Always 256 bits
More Properties of Ideal Hash Functions

We should not be able to easily find the input

Given the output:
'c74c28b6dfc5099a3ce5386ae9866fadd44618c7db489ebe87b79053131f988d'

One Way (OW):

Given y, “infeasible” to find any x such that h(x) = y.
Takeaways

- Can feed in any size data as input → fixed length output
- Randomness: Small change in input data → entirely different output
- One-way: Given a hash, can’t feasibly find any data that hashes to it
- Perfect for a tamper-evident, append-only log!
Blockchain: A Closer Look!

Block 1

\[ h_1 = H(H(TXNS_1)) \]

\[ TXNS_1 \]

**Block Header:**
- Contains hash pointer to TXN data
- Contains hash pointer to previous block
Blockchain: A Closer Look!

Block Header:
- Contains hash pointer to TXN data
- Contains hash pointer to previous block

$$h_1 = H(H(TXNS_1))$$

$$h_2 = H(h_1, H(TXNS_2))$$

$$h_1 = H(TXNS_1)$$

$$h_2 = H(h_1, TXNS_2)$$
Blockchain: A Closer Look!

Block Header:
- Contains hash pointer to TXN data
- Contains hash pointer to previous block
Transactions: A Closer Look!

- Transactions stored in *Merkle Tree*
- *Merkle Root* hash stored in header

\[
\begin{align*}
\text{A} & \rightarrow \text{C}, \text{5 BTC} \\
\text{B} & \rightarrow \text{C}, \text{10 BTC}
\end{align*}
\]
Transactions: Membership Proof

Block $n$

$h_n = H(h_{n-1}, R_n)$

$TXNS_n$

$H(tx_1)$  $H(tx_2)$  $H(tx_3)$  $H(tx_4)$

$tx_2$

A $\rightarrow$ C, 5 BTC
B $\rightarrow$ C, 10 BTC

Membership proof for $tx_2$?
Transactions: Membership Proof

Block $n$

$h_n = H(h_{n-1}, R_n)$

$TXNS_n$

Membership proof for $tx_2$

$A \rightarrow C, 5$ BTC
$B \rightarrow C, 10$ BTC

$tx_2$

$H(tx_1)$ $H(tx_2)$ $H(tx_3)$ $H(tx_4)$
Transactions: Membership Proof

A → C, 5 BTC
B → C, 10 BTC

h_n = H(h_{n-1}, R_n)

Membership proof for tx_2
Transactions: Membership Proof

Block \( n \)

\[ h_n = H(h_{n-1}, R_n) \]

TXNS\(_n\)

Membership proof for \(tx_2\)

\( A \rightarrow C, 5\) BTC

\( B \rightarrow C, 10\) BTC
Transactions: Membership Proof

\[ M_1 = H(H(tx_1), H(tx_2)) \]

Membership proof for \( tx_2 \)
Membership Proof Success

$M_1 = H(H(tx_1), H(tx_2))$

$R_n' = H(M_1, M_2)$

$h_n = H(h_{n-1}, R_n)$

$R_n' = H(M_1, M_2)$

Yay!
Transaction Format

- Merkle tree of TXNs in each block
Transactions transfer coins
Transaction Format

- Transactions transfer coins
- Output = # of coins and owner's PK
- Input "spends" previous output
Transaction Format

Data can be embedded in TXNs.
Transaction Format

Bob gives Carl 3Ƀ,
What happens if Bob tries to double spend?
No double-spent coins: A TXN output can only be referred to by a single TXN input.
Problem with Keybase - Equivocation Within a Block?

Auditors must download the entire Bitcoin blockchain to check if any blocks contain invalid Keybase transactions.
Key idea behind Catena

Efficiently use Bitcoin's mechanism that prevents double spends as proof of non-equivocation.
Key idea behind Catena

Efficiently use Bitcoin's mechanism that prevents double spends as proof of non-equivocation.
No inconsistent $s'_{3}$ as it would require a double-spend!

Linear chain of transactions containing statements
Efficient auditing

![Diagram showing Keybase server, Keychat client, and GTX connected to the Bitcoin Network.](image)
Efficient auditing

Keybase server

Keychat client

Q: Next block header(s)?

GTX

Bitcoin Network

Keybase server
Efficient auditing

Keybase server

Keychat client

GTX

Header i

Header i+1

Header j

Bitcoin Network

Keybase server
Efficient auditing

Keybase server

Keychat client

Header i

Header j

GTX

Bitcoin Network

Keybase server
Efficient auditing

Q: What is $s_i$ in the log?
Efficient auditing

Keychat client

GTX

Header i

Header j

Bitcoin Network

Keybase server

TX₁ s₁
Efficient auditing

Keybase server

Keychat client

Bitcoin Network

Keybase server
Efficient auditing

Q: Next block header(s)?

Bitcoin Network

Keybase server
Efficient auditing

Keybase server

Keychat client

GTX

TX\_1\_s\_1

header i

header j

header j+1

header n

Bitcoin Network
Efficient auditing
Efficient auditing

Q: What is $s_2$ in the log?
Efficient auditing

![Diagram showing the process of efficient auditing in a Bitcoin network with Keybase server and Keychat client.]
Efficient auditing

Keybase server

Keychat client

Bitcoin Network

Keybase server
Auditing bandwidth

e.g., 460K block headers + 10K statements = ~41 MB
(80 bytes each)  
(around 600 bytes each)
Recap

- Keybase can equivocate, so they witness the directory in Bitcoin, but inefficiently

Keybase \rightarrow S_1, S_2, S_3, ... \rightarrow Bitcoin

\rightarrow \rightarrow \rightarrow

Keychat client (110 GB)
Recap

- Keybase can equivocate, so they witness the directory in Bitcoin, but inefficiently
- Use Catena to make auditing the Keybase PKD more efficient
Outline

Keybase

Bitcoin

Catena

Keychat
Keychat

- Uses the Keybase PKD, so users can communicate securely without fear of public key equivocation
- Implemented using Meteor, a Javascript framework that allows KeyChat to work as both a website and an Android app
Next steps

- Implement Catena for Keybase using Java to efficiently witness the Keybase Public Key Directory in the Bitcoin blockchain
- Implement Keychat using Meteor
Acknowledgements

Thanks to our mentor Alin Tomescu for his support and guidance!

Thanks to PRIMES for this opportunity!

Thanks to our parents for their support!

Thanks to all of you for being such a great audience!
Ask us questions!

Keychat

Uses

Witnesses

Keybase

Optimizes

Catena

Optimizes

Witnesses

Bitcoin