Blockchain-based Decentralized Public Key Management for Named Data Networking

Kan Yang, Lan Wang
Dept. of Computer Science, University of Memphis, USA

Jobin J. Sunny
St. Jude Children's Research Hospital

Paper in Proceedings of IEEE ICCCN 2018, Hangzhou, China
Outline

• Data-Centric Security in NDN
• Public Key Management and Compromised CA Problem
• BC-PKM: Blockchain-based Decentralized Public Key Management
  • Framework
  • Concrete Design
  • Prototype
• Conclusion and Future Work
Named Data Networking

• Bind data with a name
• Retrieve data by its name
• Forward data interest directly on names
• Forward data along the interest path
Named Data Networking

Interest Packet

<table>
<thead>
<tr>
<th>Content Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nonce</td>
</tr>
<tr>
<td>Guiders (scope, lifetime)</td>
</tr>
</tbody>
</table>

Data Packet

<table>
<thead>
<tr>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>MetaInfo</td>
</tr>
<tr>
<td>Content</td>
</tr>
<tr>
<td>Signature</td>
</tr>
</tbody>
</table>
Data Centric Security

- All the content must be signed!
- Routers may verify the signature.
- Users must verify the signature.

**Data Packet**

<table>
<thead>
<tr>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>MetaInfo</td>
</tr>
<tr>
<td>Content</td>
</tr>
<tr>
<td><strong>Signature</strong></td>
</tr>
</tbody>
</table>
Signature from Public Key Cryptography

• Data signature is usually generated by the secret key of the producer
  \[ \sigma = Sign(sk, data) \]

• The signature can be validated by the public key of the producer
  \[ Verify(pk, data, \sigma) \rightarrow Success/Fail \]
Public Key Management (PKM)

• Data signature is usually generated by the secret key of the producer
  \[ \sigma = \text{Sign}(sk, data) \]
• The signature can be validated by the public key of the producer
  \[ \text{Verify}(pk, data, \sigma) \rightarrow \text{Success/Fail} \]

Impersonate Alice by signing the data with his secret key:
\[ \sigma_{alice} = \text{Sign}(sk_{eve}, data) \]
and claiming that Alice’s public key is \( pk_{eve} \).
\[ \text{Verify}(pk_{eve}, data, \sigma_{alice}) \rightarrow \text{Success/Fail} \]

Public Key Management is the foundation of the data-centric security!
Trust Schema: Current PKM in NDN

- Trust Relationship
- Recursively validate the signature until it reaches the trust anchor

The public key of Trust Anchor is wired into the hardware or preconfigured in the software.

Traditional CA-based PKI

• Certificate Authority (CA) issues a certificate to prove that the public key is associated with a name.

Compromised CA Problem

An attacker can compromise a CA to bind a name to an unauthorized public key and produce false data using the fake certificate.
## Compromised CA Incidents

<table>
<thead>
<tr>
<th>Year</th>
<th>Incidents</th>
</tr>
</thead>
<tbody>
<tr>
<td>2001</td>
<td>• VeriSign issues Microsoft Corporation code signing certificate to a non-Microsoft employee.</td>
</tr>
</tbody>
</table>
| 2008 | • Thawte issues certificate for Live.com to non-Microsoft employee  
      • Comodo issues mozilla.org certificate to Startcom  
      • Organization forges VeriSign RapidSSL certificates |
| 2011 | • Comodo issues nine counterfeit certificates (Google, Yahoo, Live, etc.) when registration authority is compromised.  
      • StartSSL CA compromised  
      • DigiNotar compromised. 531 fraudulent certificates issued.  
      • Boeing CA compromised |
| 2012 | • Microsoft CA certificates forged by exploiting MD5 (Flame) |
| 2013 | • Fraudulent certificates on Google domains issued by the French Ministry of Finance CA (ANSSI) |
| 2014 | • Intermediate CA in India compromised |
| 2015 | • Dell notebooks with rogue root CA |
| 2016 | • One CA attacked another by attempting to trademark the brands used by the second CA |

Attacks from Compromised CA

Once the CA has been compromised, the CA has superpower to

- Register public keys for illegitimate principals
- Update public keys for existing principals
- Revoke public keys for legitimate principals
Our Idea: Reduce the superpower of single CA

Once the CA has been compromised, the CA has superpower to

- Register public keys for illegitimate principals
- Update public keys for existing principals
- Revoke public keys for legitimate principals

Our idea: Replace single CA with a set of Validators with lower privileges

- do the name-principal validation
- follow the majority principle to implement the public key management functions
Solution: Decentralize CAs and Publish Their Actions

• Replace the CA at each level with a set of Validators with lower privileges
  • A validator can publish a public key record only if a majority agrees.
• Publish every public key record in a tamper-proof blockchain
• Majority rule → As long as the attacker cannot compromise half or more of the validators, an invalid public key record will not be issued.
• Tamperproofness → Even if a validator misbehaves and publishes an invalid public key record, this can be detected by other validators through the blockchain.
Overview of Public Key Management in NDN

Public Key Management

PKMiner

Register  Query  Validate  Update  Revoke

Named Data Networking
Framework of Blockchain-based PKM (BC-PKM)

PKI

- CA_root
  - pkchain
  - CAedu
    - pkchain
    - CAmemphis
      - pkchain

BC-PKM

- Name: /root/pkchain
  - PKMiners
- Name: /edu/pkchain
  - PKMiners
- Name: /edu/memphis/pkchain
  - PKMiners

Name: /edu/memphis/pkchain

PKMiners

- Timestamp
- Nonce
- Hash of Previous Block
- Merkle Root
- Name-PK Records
  - (alice, pkAlice)
  - (bob, pkBob)
  - (charlie, pkCharlie)

APIs:
- Register
- Query
- Validate
- Update
- Revoke

Name: /edu/memphis/alice/data

Content:

Signature:
- KeyLocator:
  - /edu/memphis/pkchain/query/alice

Name: /edu/memphis/pkchain/query/alice

The University of Memphis
PKChain: Register

Name: /edu/memphis/pkchain

1. Broadcast (alice, pk_{alice})

2. Name-Principal Validation

3. Majority Accepted

4. Packet & Min

5. Broadcast New Block

Register API

PKMiners

Producer

Timestamp Nonce
Hash of Previous Block
Merkle Root
Name-PK Records (alice, pk_{alice})
PKChain: Query

1. Send Query \((\text{alice, pk}_{\text{alice}})\) to a random PKMiner

2. Search from the last block towards the first block

3. Return the first record (the latest record) or NotFound

Query API

Name: /edu/memphis/pkchain

/{edu/memphis/pkchain/query/\text{alice} \text{or pk}_{\text{alice}})

By name or by public key
PKChain: Validate (query and compare)

1. Send Validate(alice, pk_{alice}) to a random PKMiner

2. Search from the last block towards the first block

3. Return the first record (the latest record) and compare with the submitted pair

Validate API

Name: /edu/memphis/pkchain

PKMiners

/edu/memphis/pkchain/validate/(alice, pk_{alice})
PKChains: Update (add a new block)
PKChain: Revoke (add a revoking block)

Name: /edu/memphis/pkchain

PKMiners

1. Broadcast Revoke\((alice, pk_{alice})\)

2. Name-Principal Validation

Produce

3. Majority Accepted

4. Packet & Min

5. Broadcast New Block

Merkle Root

Hash of Previous Block

Name-PK Records
\((alice, pk_{alice}, revoked)\)

Timestamp

Nonce
Security Analysis

Theorem: BC-PKM can resist \( t \) out of \( n \) (\( n > 2t - 1 \)) compromised PKMiners against
- registering public keys for fake principals
- illegally updating public keys for existing principals
- illegally revoking public keys for existing principals

as long as there are more than half PKMiners are honest.

Guaranteed by the majority principle of the name-principal validation.

Refer to the paper for more details.
Prototype

- Implement by the Node.js framework
  - due to its asynchronous capabilities and ability to handle peer-to-peer communications well
  - The framework’s event-driven, non-blocking I/O model makes it a good fit for our implementation.

- The command line interface was created with the help of a library called Vorpal (https://www.npmjs.com/package/vorpal)

- Note that, the main purpose of this prototype is to validate the functions of our BC-PKM system
**Prototype: Register**

<table>
<thead>
<tr>
<th>Timestamp</th>
<th>Mon, 06 Nov 2017 17:35:48 GMT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Blockchain PKI</td>
</tr>
<tr>
<td>Public Key</td>
<td>Ø</td>
</tr>
<tr>
<td>Hash</td>
<td>0000849f7250903ea57f1614e1d16fc750a6c...</td>
</tr>
<tr>
<td>Nonce</td>
<td>113708</td>
</tr>
<tr>
<td>Miner ID</td>
<td>-1</td>
</tr>
<tr>
<td>Revoked</td>
<td>false</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Block #1</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Previous Hash</td>
<td>0000849f7250903ea57f1614e1d16fc750a6c...</td>
</tr>
<tr>
<td>Timestamp</td>
<td>Tue, 17 Apr 2018 23:43:28 GMT</td>
</tr>
<tr>
<td>Name</td>
<td>John</td>
</tr>
<tr>
<td>Public Key</td>
<td>1xui45nal</td>
</tr>
<tr>
<td>Hash</td>
<td>000043792502902acf2839b11d39508548fa9...</td>
</tr>
<tr>
<td>Nonce</td>
<td>40951</td>
</tr>
<tr>
<td>Miner ID</td>
<td>0</td>
</tr>
<tr>
<td>Revoked</td>
<td>false</td>
</tr>
</tbody>
</table>

**Block #1**

- **Previous Hash**: 0000849f7250903ea57f1614e1d16fc750a6c...
- **Timestamp**: Tue, 17 Apr 2018 23:43:28 GMT
- **Name**: John
- **Public Key**: 1xui45nal
- **Hash**: 000043792502902acf2839b11d39508548fa9...
- **Nonce**: 40951
- **Miner ID**: 0
- **Revoked**: false

**BC-PKMS$ register Mary bqnas83p2a3iz
Congratulations! A new block was mined.**

**BC-PKMS$**
Prototype: Query then Update

<table>
<thead>
<tr>
<th>Name</th>
<th>John</th>
<th>PKMiner1</th>
<th>PKMiner2</th>
<th>PKMiner3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Public Key</td>
<td>1xu145nal</td>
<td>Timestamp</td>
<td>Timestamp</td>
<td>Timestamp</td>
</tr>
<tr>
<td>Hash</td>
<td>00007effc506666c3303f8e37ecbf0fa4ca12...</td>
<td>Tue, 17 Apr 2018 23:59:57 GMT</td>
<td>Name: Mary, Public Key: bqnas83p2a3iz</td>
<td></td>
</tr>
<tr>
<td>Nonce</td>
<td>23204</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Miner ID</td>
<td>Φ</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Revoked</td>
<td>false</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Block #2

<table>
<thead>
<tr>
<th>Name</th>
<th>Mary</th>
<th>PKMiner1</th>
<th>PKMiner2</th>
<th>PKMiner3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Public Key</td>
<td>bqnas83p2a3iz</td>
<td>Timestamp</td>
<td>Timestamp</td>
<td>Timestamp</td>
</tr>
<tr>
<td>Hash</td>
<td>0000ec4f9e9f19dc5c79719643d72b6598db2...</td>
<td>Tue, 17 Apr 2018 23:59:57 GMT</td>
<td>Name: Mary, Public Key: bqnas83p2a3iz</td>
<td></td>
</tr>
<tr>
<td>Nonce</td>
<td>137170</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Miner ID</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Revoked</td>
<td>false</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

BC-PKMS  |  |  |  |  |
Prototype: Updated View then Revoke

<table>
<thead>
<tr>
<th>Name</th>
<th>Mary</th>
<th>PKMiner1</th>
<th>PKMiner2</th>
<th>PKMiner3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Public Key</td>
<td>bqnas83p2a31z</td>
<td>Timestamp</td>
<td>Name</td>
<td>Name</td>
</tr>
<tr>
<td>Hash</td>
<td>0000ec4f9e9f19dc5c79719643d72b6598db2...</td>
<td>Tue, 17 Apr 2018 23:59:57 GMT</td>
<td>Mary</td>
<td>bqnas83p2a31z</td>
</tr>
<tr>
<td>Nonce</td>
<td>137170</td>
<td></td>
<td>Public Key</td>
<td>Public Key</td>
</tr>
<tr>
<td>Revoked</td>
<td>false</td>
<td></td>
<td>Hash</td>
<td>Hash</td>
</tr>
<tr>
<td>Block #3</td>
<td></td>
<td></td>
<td>Nonce</td>
<td>Nonce</td>
</tr>
<tr>
<td>Previous Hash</td>
<td>0000ec4f9e9f19dc5c79719643d72b6598db2...</td>
<td></td>
<td>Miner ID</td>
<td>Miner ID</td>
</tr>
<tr>
<td>Timestamp</td>
<td>Wed, 18 Apr 2018 00:01:40 GMT</td>
<td></td>
<td>Revoked</td>
<td>Revoked</td>
</tr>
<tr>
<td>Name</td>
<td>John</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Public Key</td>
<td>7fhqnl3sg8gb</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hash</td>
<td>00004c2840e1a810372ce4cf553106ba790f8...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nonce</td>
<td>134724</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Miner ID</td>
<td>0</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Revoked</td>
<td>false</td>
<td></td>
<td></td>
<td>false</td>
</tr>
</tbody>
</table>

BC-PKMS $ queryname John
Name: John, Public Key: 1xui45n1
BC-PKMS $ querypk bqnas83p2a31z
Name: Mary, Public Key: bqnas83p2a31z
BC-PKMS $ querypk bqnas83p2a31z
Name: Mary, Public Key: bqnas83p2a31z
BC-PKMS $ querypk 1xui45n1
Name: John, Public Key: 1xui45n1
BC-PKMS $ querypk 1xui45n1
Name: John, Public Key: 1xui45n1
The queried public key is no longer valid
BC-PKMS $ querypk 7fhqnl3sg8gb
Name: John, Public Key: 7fhqnl3sg8gb
BC-PKMS $
Prototype: Revoked View and Validate

<table>
<thead>
<tr>
<th>Timestamp</th>
<th>Wed, 18 Apr 2018 01:47:47 GMT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>John</td>
</tr>
<tr>
<td>Public Key</td>
<td>7fhqn13sg8gb</td>
</tr>
<tr>
<td>Hash</td>
<td>0000c95be54c8450f267a90784c7861e92765...</td>
</tr>
<tr>
<td>Nonce</td>
<td>12158</td>
</tr>
<tr>
<td>Miner ID</td>
<td>0</td>
</tr>
<tr>
<td>Revoked</td>
<td>false</td>
</tr>
</tbody>
</table>

**Block #4**

<table>
<thead>
<tr>
<th>Previous Hash</th>
<th>0000c95be54c8450f267a90784c7861e92765...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timestamp</td>
<td>Wed, 18 Apr 2018 01:47:59 GMT</td>
</tr>
<tr>
<td>Name</td>
<td>John</td>
</tr>
<tr>
<td>Public Key</td>
<td>7fhqn13sg8gb</td>
</tr>
<tr>
<td>Hash</td>
<td>0000c1e867849c79b9be71d242971ba128f1c...</td>
</tr>
<tr>
<td>Nonce</td>
<td>137632</td>
</tr>
<tr>
<td>Miner ID</td>
<td>0</td>
</tr>
<tr>
<td>Revoked</td>
<td>true</td>
</tr>
</tbody>
</table>

BC-PKM$ validate John 7fhqn13sg8gb
This pair is not valid.
BC-PKM$ validate Mary bqnas83p2a3iz
The pair is valid.
BC-PKM$ validate Mary 123456
This pair is not valid.
BC-PKM$
Conclusion and Future Work

• We proposed BC-PKM, a blockchain-based decentralized public key management system, for Named Data Networking.

• The BC-PKM can solve the compromised CA problem existing in traditional PKM systems and can tolerate less than half PKMiners are compromised by the adversary while keeping the system stable and secure.

In our future work, we will solve the following design questions:

• Who can be the miners/validators?
• How to validate a public key? How to do the name-principal validation?
• Which consensus mechanism should we use?
Thank You!

kan.yang@memphis.edu