CONIKS (KEY TRANSPARENCY)

Slides adapted from Marcela Melara
Any scriber for today?

- Sign up for presentations or scribers
The problem of the PKI (Public key infrastructure)

• A long standing problem has been to distribute public keys securely in the presence of attackers

• Use cases:
  • Secure messaging: Alice needs to send an encrypted message to Bob and needs Bob’s public key to encrypt the message
  • Web surfing and https: when Alice’s browser contacts amazon.com over https, we need Amazon’s PK
  • Man in the middle attacker can intercept PK and return incorrect PK
“Secure” Communication Today

Email Provider

`foo.com`

Certificate Authority

SSL/TLS Certificate: `(foo.com, PK_f)`

User

_Alice_

User

_Bob_

HTTP + SSL/TLS
Attack: Hackers

Email Provider: foo.com

Certificate Authority

User Alice

User Bob
Attack: Disgruntled Employees

Email Provider "foo.com"

Certificate Authority

Certificate: (foo.com, PK_f)

User Alice

HTTP + SSL/TLS

User Bob

HTTP + SSL/TLS
Attack: Provider Coercion

Email Provider *foo.com*

Certificate Authority

SSL/TLS Certificate: *(foo.com, PK_f)*

User *Alice*

User *Bob*
CAs are Vulnerable

Email Provider  
\textit{foo.com}

Certificate Authority

SSL/TLS Certificate:  
\langle \text{foo.com}, \text{PK}_f \rangle

User  
\textit{Alice}

User  
\textit{Bob}
CAs are Vulnerable

Email Provider  
foo.com

Certificate Authority

SSL/TLS Certificate:  
(foo.com, PK_f)

HTTP + SSL/TLS

User Alice

HTTP + SSL/TLS

User Bob

SSL/TLS

“hey, I’m foo.com”

Hacker  
foo.com
CAs are Vulnerable
Attack: Fraudulent Certificates

Email Provider

(foo.com)

Certificate Authority

SSL/TLS Certificate: (foo.com, PK_{f})

User

Alice

User

Bob

HTTP + SSL/TLS

HTTP + SSL/TLS
Attack: Fraudulent Certificates

- Email Provider: *foo.com*
- Certificate Authority
  - Certificate: *(foo.com, PK_f)*
  - SSL/TLS
- Hacker: *foo.com*
  - Certificate: *(foo.com, PK_{evil})*
- User: *Alice*
  - HTTP + SSL/TLS
- User: *Bob*
  - HTTP + SSL/TLS
Need End-to-End Encryption…

Email Provider
foo.com

User Alice

User Bob

… But Key Management is Hard
Decentralized Key Management

User Alice

PK_{Alice}: DEF456

User Bob

PK_{Bob}: 123ABC

Manual key exchange
Decentralized Key Management

User Alice trusts PK\textsubscript{Bob}

Bob trusts PK\textsubscript{Alice}

Mutual Endorsement
Decentralized KM: Pitfalls

User Alice: DEF456

PK_{Alice}: DEF456

PK_{Bob}: 23ABC

User Bob

Lost keys
Decentralized KM: Pitfalls

Mistakes transferring keys
PGP Key Servers

Email Provider
foo.com

User Alice

User Bob

PGP Key Server X

E2E Encryption

PK_{Bob} → PGP Key Server X

PK_{Alice} → PGP Key Server X
## PGP Key Servers

### Search results for 'melara marcela'

<table>
<thead>
<tr>
<th>Type</th>
<th>bits/keyID</th>
<th>Date</th>
<th>User ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>pub</td>
<td>4096R/99AAFE63</td>
<td>2016-04-08</td>
<td>Marcela S. Melara <a href="mailto:msmelara@gmail.com">msmelara@gmail.com</a></td>
</tr>
<tr>
<td>pub</td>
<td>2048R/4AF7855E</td>
<td>2016-01-18</td>
<td>Marcela S. Melara <a href="mailto:melara@cs.princeton.edu">melara@cs.princeton.edu</a></td>
</tr>
<tr>
<td>pub</td>
<td>4096R/CDC3499E</td>
<td>2015-04-11</td>
<td>*** KEY REVOKED *** [not verified] Marcela Melara <a href="mailto:melara@cs.princeton.edu">melara@cs.princeton.edu</a></td>
</tr>
<tr>
<td>pub</td>
<td>2048R/69B2E80C</td>
<td>2012-10-01</td>
<td>Marcela Melara (test) <a href="mailto:msmelara@gmail.com">msmelara@gmail.com</a></td>
</tr>
</tbody>
</table>
Key Signing Parties
Trusting the Web of Trust

Hey, I just got home from the party. The one with the IRC folks?

Yeah.

How was it?

Got too drunk. I screwed up, bad.

What happened?

There was a girl. No idea who she was. Don't even know her name. I was too drunk to care.

And what, you slept with her?

No.

I signed her public key.

Shit, man.

XKCD, Responsible Behavior
Crux of WoT Issues

• Decentralized model: people reason about encryption.

• Studies:
  • Unintuitive and error-prone.
  • Users don’t understand encryption.
  • Leak private information.

Better: Centralized Key Management

Register (alice → PK_A)

User Alice

Register (bob → PK_B)

User Bob

Email + Key Provider
foo.com
Better: Centralized Key Management

Email + Key Provider
foo.com

1. Register (alice → PKₐ)
2. Look up Alice’s public key: PKₐ
3. Send message encrypted to PKₐ, signed by SKₖ

User Alice

User Bob
Insider Attacks, still.

1. Register (alice → PKₐ)
2. Look up Alice’s public key: PK’ₐ

Email + Key Provider foo.com

This isn’t Alice’s real key!
Insider Attacks, still.

Email + Key Provider *foo.com*

1. Email + Key Provider *foo.com*
2. Read message encrypted to PK'ₐ
3. Send message encrypted to PK'ₐ, signed by SKₐ
4. User Alice
5. User Bob
Old Approach: Correct Identities

Alice
alice@foo.com

Certificate
Name: alice@foo.com
PKₐ: 456DEF
Owner: Alice
Signed by: CA

Bob’s real-world friend Alice?
New Approach: Consistent Identities

• Users expect consistency of online identities.

• Separate real-world identity from online identity.

• Certificates make no statement about correctness.
Consistency

No unexpected key changes

Alice’s key today = Alice’s key yesterday

Non-Equivocation

Key seen by Alice = Key seen by Bob
Solution: a promising new generation

- CONIKS
  - An alternative Public Key Infrastructure (PKI)
  - Embedded in Google’s project called Key Transparency or Trillian
    - The key data structure is the Log-backed Verified Map

- Certificate Transparency
  - Similar technology to Key Transparency but aimed at certificates
  - Currently mandated by Chrome, other browser support coming up
CONIKS [Melara et al. 2014]

- End-User Key Management Service.
- Consistent name-to-key bindings.
- Verifiable key directories \rightarrow Untrusted identity providers.
- Clients verify consistency in-band.
  \rightarrow automated key management
CONIKS Overview

User Alice

Register (alice \(\rightarrow\) PK\(_A\))

Client A

Untrusted Identity Provider \textit{foo.com}

Register (bob \(\rightarrow\) PK\(_B\))

Client B

User Bob
CONIKS Overview

Untrusted Identity Provider
foo.com

1. Look up the public key for Alice: PK_A
2. Look up the public key for Bob: PK_B
3. Send message encrypted to PK_A, signed by SK_B
4. Decrypt using SK_A
Strawman Design

Validity Checks
$O(N)$ storage per client

Non-equivocation Checks
$O(N^2)$ downloads per client

Untrusted Identity Provider
foo.com

Client A
Client B
Client C
Client D

$N = 4$
CONIKS Design

- Divide time into epochs.

- Providers generate snapshots of directory.
  - Clients do not check individual bindings.

- Providers distribute snapshots to other providers.
  - Build publicly verifiable history

- Non-repudiation: Snapshots are digitally signed.
Efficient Snapshots

\[ H(\text{sub}_L) \quad H(\text{sub}_R) \]

root

\[ i_{alice} : PK_{Alice} \]

\[ i_{charlie} : PK_{Charlie} \]

\[ i_{emily} : PK_{Emily} \]

\[ i_{george} : PK_{George} \]
Efficient Snapshots

$H(\text{sub}_L)$  $H(\text{sub}_R)$

$H(\text{sub}_L)$  $H(\text{sub}_R)$

$H(\text{sub}_L)$  $H(\text{sub}_R)$

$H(\text{sub}_L)$  $H(\text{sub}_R)$

$i_{alice} : PK_{Alice}$  $i_{charlie} : PK_{Charlie}$  $i_{emily} : PK_{Emily}$  $i_{george} : PK_{George}$

$foo.com$
Checking Consistency

- Use snapshots to check for consistency.
- Clients need to ensure bindings are included in snapshots.
  - Lookups: prove inclusion of bindings in directory.
- Clients check validity, non-equivocation.
- Providers cross-verify each other for non-equivocation.
Proving Inclusion: Authentication Paths

foo.com

H(sub_L)  H(sub_R)

root

H(sub_L)  H(sub_R)

0  1

0  1

i_{alice}: PK_{Alice}

i_{charlie}: PK_{Charlie}
Proving Inclusion: Authentication Paths

\[ i_{alice} : PK_{Alice} \]

\[ \text{root} \]

\[ H(\text{sub}_L) \]

\[ H(\text{sub}_R) \]

\[ \text{foo.com} \]
Checking Non-Equivocation: Snapshot History

Trillian calls this: Log-backed Verified Map
Checking Non-Equivocation: Snapshot History

The server can try a fork attack, but after fork the provider must maintain these forked hash chains for the rest of time, and not allow clients seeing one branch of the hash chain to communicate with anyone seeing the other branch.
CONIKS Protocols

- Registration: New bindings/updated keys.
- Lookups: Clients check bindings are included in directory.
- Monitoring: Clients check for validity of bindings.
- Auditing: Clients & providers check for non-equivocation.
CONIKS Protocols

- **Registration**: New bindings/updated keys.
- **Lookups**: Clients check bindings are included in directory.
- **Monitoring**: Clients check for validity of bindings.
- **Auditing**: Clients & providers check for non-equivocation.
Registration Protocol

1. User Alice generates key pair \((PK_A, SK_A)\).
2. User Alice registers with the Untrusted Identity Provider \(foo.com\) (alice \(\rightarrow\) PK\(_A\)).
3. Temporary binding = \([(alice \rightarrow PK_A) + \text{next epoch}], \text{Sig}(TB)\).
CONIKS Protocols

• Registration: new bindings/updated keys.

• Lookups: Clients check bindings are included in directory.

• Monitoring: Clients check for validity of bindings.

• Auditing: Clients & providers check for non-equivocation.
Lookups without Inclusion Proofs

- User Alice
- Provider regularly generates and publishes snapshots
  - Register (alice $\rightarrow$ PK$_A$)
  - Untrusted Identity Provider *foo.com*
  - Publish new snapshot including PK$_A$
- Client A
- Client B
- User Bob

Provider regularly generates and publishes snapshots
Lookups without Inclusion Proofs

Untrusted Identity Provider \textit{foo.com}

This isn't alice's real key!

Look up the public key for alice: PK'_A

Return Fake Key
Lookups without Inclusion Proofs

Untrusted Identity Provider \textit{foo.com}

No proof fake key is inconsistent with snapshot
Lookups without Inclusion Proofs

Untrusted Identity Provider
foo.com

User Alice

Client A

Read message encrypted to PK'_A

Provider can read Bob’s message

Send message encrypted to PK'_A, signed by SK_B

Client B

User Bob
Lookup Protocol

- Ensures bindings are consistent with snapshots.
- Prevents malicious providers from publishing fake keys and not leaving any evidence of the misbehavior.
Lookup Protocol

1. Lookup (alice $\rightarrow$ PK$_A$)
2. Send proof of inclusion: Authentication path for (alice $\rightarrow$ PK$_A$)
3. Verify auth. path
CONIKS Protocols

• Registration: new bindings/updated keys.

• Lookups: Clients check bindings are included in directory.

• Monitoring: Clients check for validity of bindings.

• Auditing: Clients & providers check for non-equivocation.
Communication without Monitoring

Untrusted Identity Provider `foo.com`

Register (alice → PKₐ)

Look up the public key for alice: PKₐ

User Alice

Client A

Client B

User Bob

Epoch 1: Key has not Changed
Communication without Monitoring

Untrusted Identity Provider
foo.com

Send message encrypted to PK_A, signed by SK_B

User Alice

Client A

Untrusted Identity Provider
foo.com

User Bob

Client B

Epoch 1: Provider cannot read Bob’s message
Communication without Monitoring

Untrusted Identity Provider \textit{foo.com}

Look up the public key for alice: $\text{PK}'_A$

This isn't Alice's real key!

Epoch 2: Key has been Changed
Communication without Monitoring

Untrusted Identity Provider *foo.com*

Read message encrypted to $PK'_A$

Send message encrypted to $PK'_A$, signed by $SK_B$

User Alice

Client A

User Bob

Client B

Epoch 2: Provider can read Bob’s message
Monitoring Protocol

• Ensures public keys do not change unexpectedly.

• Prevents malicious providers from getting away with replacing existing keys.
Monitoring Protocol

1. Lookup (alice $\rightarrow$ PK$_A$)
2. Send (alice $\rightarrow$ PK$_A$) + authentication path
3. Verify validity of binding & auth. path
CONIKS Protocols

- Registration: new bindings/updated keys.

- Lookups: Clients check bindings are included in directory.

- Monitoring: Clients check for validity of bindings.

- Auditing: Clients & providers check for non-equivocation.
Communication without Auditing

Clients register legitimate Keys

User Alice

Client A

Untrusted Identity Provider foo.com

Register (alice → PK_A)

Register (bob → PK_B)

User Bob

Client B
Communication without Auditing

Untrusted Identity Provider
foo.com

Provider creates two versions of its Directory

User Alice

Client A

client A sees PK'_{B} as bob’s public key

Client B

client B sees PK'_{A} as alice’s public key
Communication without Auditing

Provider presents two different but valid snapshots
Auditing Protocol

- Ensures clients have consistent views of directories.
- Clients query random other providers for snapshots observed from given provider.
- Prevents malicious providers from getting away with publishing inconsistent versions of key directory.
Auditing Protocol

1. Verify foo.com’s snapshot history
   - Untrusted Identity Provider: rando.com

2. Distribute signed snapshot
   - Untrusted Identity Provider: foo.com

3. Verify foo.com’s snapshot history
   - Untrusted Identity Provider: bar.com

User: Alice

Clients: A, B
Auditing Protocol

1. Verify foo.com’s snapshot history

Untrusted Identity Provider
rando.com

2. Request most recent snapshot

Untrusted Identity Provider
foo.com

3. Send most recent snapshot

Untrusted Identity Provider
bar.com

4. Verify foo.com’s snapshot history

Client
B

User
Alice

Client
A
Checking Snapshot History

Check signature on snapshot → Valid → Compare $H(root_{prev})$ of cached and new snapshot → Match → Check passed
Checking Snapshot History

Check signature on snapshot → Valid

Compare $H(root_{prev})$ of cached and new snapshot

Not matching → Fail
Auditing Protocol

User Alice

Client A

Request snapshot observed for foo.com

Request snapshot observed for foo.com

Client B

Compare observed snapshots for foo.com

Untrusted Identity Provider rando.com

Untrusted Identity Provider foo.com

Untrusted Identity Provider bar.com
Privacy problems with this protocol?

- Auditors see contents of directory
- Authentication paths reveal other entries in the directory
Prototype Implementation

- CONIKS Chat: Secure chat service.
- Stand-alone provider alongside XMPP server.
- Supports registration, lookups and basic auditing.
Server performance

Time for 1k updates (s)

Number of existing users

- $1 \times 10^5$
- $1 \times 10^6$
- $1 \times 10^7$
Client Overheads

- \( N \approx 4 \text{ bill. users}, \ n \approx 2 \text{ mill. updates/epoch}, \ d = 24 \text{ epochs/day}. \)

<table>
<thead>
<tr>
<th></th>
<th>Download Requirements</th>
<th>Storage Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lookup (per binding)</td>
<td>&lt; 1.4KB</td>
<td>0B</td>
</tr>
<tr>
<td>Monitoring (epoch)</td>
<td>&lt; 800B</td>
<td>~ 300B</td>
</tr>
<tr>
<td>Monitoring (day)</td>
<td>&lt; 20KB</td>
<td>~ 300B</td>
</tr>
<tr>
<td>Auditing (epoch, per snapshot)</td>
<td>~ 100B</td>
<td>~ 100B</td>
</tr>
<tr>
<td>Auditing (day, per snapshot)</td>
<td>~ 2.5KB</td>
<td>~ 100B</td>
</tr>
</tbody>
</table>
### Client Overheads

- $N \approx 4$ bill. users, $n \approx 2$ mill. updates/epoch, $d = 24$ epochs/day.

<table>
<thead>
<tr>
<th></th>
<th>Download Requirements</th>
<th>Storage Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lookup (per binding)</td>
<td>&lt; 1.4KB</td>
<td>0B</td>
</tr>
<tr>
<td>Monitoring (epoch)</td>
<td>&lt; 800B</td>
<td>~ 300B</td>
</tr>
<tr>
<td>Monitoring (day)</td>
<td>&lt; 20KB</td>
<td>~ 300B</td>
</tr>
<tr>
<td>Auditing (epoch, per snapshot)</td>
<td>~ 100B</td>
<td>~ 100B</td>
</tr>
<tr>
<td>Auditing (day, per snapshot)</td>
<td>~ 2.5KB</td>
<td>~ 100B</td>
</tr>
</tbody>
</table>
Client Overheads

- \( N \approx 4 \text{ bill. users}, \ n \approx 2 \text{ mill. updates/epoch}, \ d = 24 \text{ epochs/day}. \)

<table>
<thead>
<tr>
<th></th>
<th>Download Requirements</th>
<th>Storage Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lookup (per binding)</td>
<td>&lt; 1.4KB</td>
<td>~ 300B</td>
</tr>
<tr>
<td>Monitoring (epoch)</td>
<td>&lt; 800B</td>
<td>~ 300B</td>
</tr>
<tr>
<td>Monitoring (day)</td>
<td>&lt; 20KB</td>
<td>~ 300B</td>
</tr>
<tr>
<td>Auditing (epoch, per snapshot)</td>
<td>~ 100B</td>
<td>~ 100B</td>
</tr>
<tr>
<td>Auditing (day, per snapshot)</td>
<td>~ 2.5KB</td>
<td>~ 100B</td>
</tr>
</tbody>
</table>
Client Overheads

- \( N \approx 4 \text{ bill. users}, \ n \approx 2 \text{ mill. updates/epoch}, \ d = 24 \text{ epochs/day} \)
CONIKS summary

- CONIKS is an efficiently verifiable, privacy-preserving key management service for end-user public keys.

- End-to-end security can be practical for end-users.

- Some industry adoption: e.g. Google’s Key Transparency