E2e-encrypted email via enhanced certificate transparency

Mark D. Ryan
School of Computer Science
University of Birmingham

Network and Distributed System Security Symposium
25 February 2014
The age of “electronic mail” may soon be upon us... (1978)

Attackers:
- Governments and security agencies
- Corporations whose business model is to monetise our data
## End-to-end encrypted mail

<table>
<thead>
<tr>
<th>S/MIME</th>
<th>OpenPGP</th>
</tr>
</thead>
</table>
| CAs certify users’ public keys  
  - Costly  
  - Messy to set up  
  - Insecure  
| Implemented in Outlook, Thunderbird, iOS Mail, OSX,... |
| Users certify each others keys: web of trust; key-signing parties  
  - Hard to understand  
  - Messy to set up  
| Not so widely implemented (there’s an extension for Thunderbird) |

Mark Ryan
End-to-end encrypted mail

S/MIME
- CAs certify users’ public keys
  - Costly
  - Messy to set up
- Implemented in Outlook, Thunderbird, iOS Mail, OSX...

OpenPGP
- Users certify each others' keys: web of trust, key-signing
  - Hard to understand
  - Messy to set up
- Not so widely implemented (there’s an extension for Thunderbird)
Major deployment obstacle:
Public key management
CT-Mail

Goal
End-to-end encrypted mail usable by people who don’t want to know anything about keys and certificates

- Certificates are managed using **certificate transparency**
  - extended to handle certificate revocations
  This allows the untrusted mail provider to act as CA
- Mail provider *proves* that it manages the keys correctly
  - Mail client software *checks the proofs*
Hi Joe, Bob's away on business.
**Aim:** ensure that whenever a CA signs a certificate, there is persistent evidence of this fact. A CA cannot sign certificates inadvertently/sneakily.

**Mechanism:** a certificate is accepted only if it is included in the *append-only public log* of certificates issued by the given CA.

The certificate comes with proof that it is included in the log. Users’ client software checks that log is *append-only* and *linear*.

**Status:** IETF draft; RFC; being implemented in Chrome.
Certificate transparency: append-only public log

\[ h(h(c_1, c_2), h(c_3, c_4), h(c_5, c_6)) \]

Algorithm | Complexity | Typical size  
--- | --- | ---
request_h() | \(O(1)\) | \(10^9\) certif.
prove_presence(h, cert) | \(O(\log n)\) |
prove_absence(h, cert) | \(O(n)\) |
prove_extension(h_1, h_2) | \(O(\log n)\) |
Certificate transparency: append-only public log

\[ h(h(h(c_1, c_2), h(c_3, c_4)), h(c_5, c_6)) \]

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Complexity</th>
<th>Typical size</th>
</tr>
</thead>
<tbody>
<tr>
<td>request_h()</td>
<td>(O(1))</td>
<td>0.25 KB</td>
</tr>
<tr>
<td>prove_presence((h), cert)</td>
<td>(O(\log n))</td>
<td>2 KB</td>
</tr>
<tr>
<td>prove_absence((h), cert)</td>
<td>(O(n))</td>
<td>60 GB</td>
</tr>
<tr>
<td>prove_extension((h_1), (h_2))</td>
<td>(O(\log n))</td>
<td>2 KB</td>
</tr>
</tbody>
</table>

Mark Ryan
Key revocation

- Cert. transp. doesn’t support proofs of absence
  - Therefore it does not support key revocation: \( \text{current} = \text{present} \land \neg \text{revoked} \)

- But we have to support revocation: lost/forgotten passwords, compromised keys, hacked accounts, . . .

- Technical challenge: extend CT to support efficient proofs of absence

- Other interesting uses for proofs of absence:
  - Incentivise deployment of CT
  - Build mechanisms to prevent TLS stripping
Proofs of currency or absence

Arrange as binary search tree, with $d_i = (subj_i, cert_i)$:
Proofs of currency or absence

Arrange as binary search tree, with $d_i = (subj_i, cert_i)$:

```
d_8
  h(d_8, h(d_4, h(d_2, h(d_1), h(d_3))), h(d_6, h(d_5), h(d_7))), h(d_{10}, h(d_9), h(d_{11})))

  d_4
  h(d_4, h(d_2, h(d_1), h(d_3))), h(d_6, h(d_5), h(d_7)))

  d_10
  h(d_{10}, h(d_9), h(d_{11}))

  d_2
  h(d_2, h(d_1), h(d_3))

  d_6
  h(d_6, h(d_5), h(d_7))

  d_9
  h(d_9)

  d_11
  h(d_{11})
```

Mark Ryan
Proof of absence

To prove there is no key for $subj$, the log maintainer provides:

- proof of presence for $subj_1$;
- proof of presence for $subj_2$;
- proof that $subj_1$ and $subj_2$ are neighbours;

Client verifies the proofs, and also that $subj_1 < subj < subj_2$ lexicographically.
Certificate Issuance and Revocation Transparency

<table>
<thead>
<tr>
<th>Proof of</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>presence</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>absence</td>
<td>$O(n)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>extension</td>
<td>$O(\log n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>consistency</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
</tbody>
</table>
Consistency checking

Two ways to check ChronTree/LexTree sync:

- **Total**: receive all updates, and check everything.
- **Random**: user client software specifies random \((c_i, l_i)\), and requests proof that \(LT(l_i) = LT(l_{i-1}) + c_i\).
Coverage of random checking

\( n \) number of users

\( \nu \) proportion of ‘victims’

(\( \text{CA is cheating about their certificates} \))

\( t \) time in days until detection with probability 0.5

\[ t = 0.1 \text{ days} \]
Alice signs up

- Application fetches current $h$ and stores it.
- Alice enters user-name “alice@example.com”, chooses new password $pw$. The software chooses an encryption key $k$.
- Alice creates public key pair $pk_{Alice}, sk_{Alice}$.
- Application stores $(Alice, \{h, pk_{Alice}, sk_{Alice}, \ldots\}_k)$ on server.
Alice sends E-mail to Bob

- Alice’s app fetches current $h'$.
- App retrieves locally stored $h_s$ and requests and verifies proof that $h_s \sqsubseteq h'$.
- App requests & verifies proof that $pk_{Alice}$ is current in $h'$.
- App authenticates Alice and fetches $(Alice, \{h, pk_{Alice}, sk_{Alice}, \ldots \}_k)$.
- App requests & verifies proofs that $h_s \sqsubseteq h \sqsubseteq h'$, and replaces $h$ and $h_s$ with $h'$.
- App requests $pk_{Bob}$ & verifies currency proof in $h'$.
- App encrypts message for Bob with $pk_{Bob}$.
<table>
<thead>
<tr>
<th>Realities of email</th>
<th>How handled</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiple devices</td>
<td>Store keys in ${\text{keypurse}}_k$ in cloud</td>
</tr>
<tr>
<td></td>
<td>Enroll new device by transferring $k$</td>
</tr>
<tr>
<td></td>
<td>Verify $h_s \subseteq h \subseteq h'$</td>
</tr>
<tr>
<td>Plaintext compat.</td>
<td>UI informs of encr. status</td>
</tr>
<tr>
<td>Webmail</td>
<td>OSS browser extension</td>
</tr>
<tr>
<td>Search</td>
<td>Restrict it to headers</td>
</tr>
<tr>
<td></td>
<td>Optionally, store $\text{HMAC}_k$(word)</td>
</tr>
<tr>
<td>Metadata prot’n, OTR</td>
<td>Not realities</td>
</tr>
<tr>
<td>Realities of email</td>
<td>Remark</td>
</tr>
<tr>
<td>---------------------------------------------------</td>
<td>---------------------------------------------</td>
</tr>
<tr>
<td>Password forgotten</td>
<td>Usual methods</td>
</tr>
<tr>
<td>Password compromised</td>
<td>Usual methods</td>
</tr>
<tr>
<td>$k$ “forgotten”</td>
<td>Lose store; reset account</td>
</tr>
<tr>
<td>$k$ compromised</td>
<td>Past email may be compr. Revoke pk; reset account</td>
</tr>
</tbody>
</table>
Why do you want end-to-end encrypted mail?

Drugs, guns, paedophilia

- You need to prevent attacks, not just detect them
- You should consider your provider to be malicious
- **CT-Mail can’t help you**

Avoid pervasive surveillance

- Detection of attacks after the event is enough
- You can consider your provider to be malicious but cautious
- **CT-Mail is for you**

Targeted attacks will bypass e2e encryption (e.g., malware, device theft, rubber hose)

Mark Ryan
Attacker models

- Malicious
- Malicious but cautious
- Honest but curious, semi-honest
- Honest
Certificate transparency

Certificate issuance and revocation transparency (CIRT)

CT-Mail
  - Usability.

Malicious-but-cautious attacker
  - Applications
  - Formalisation
  - Analysis/verification