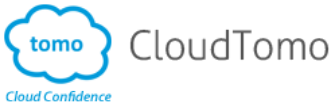
# E2e-encrypted email via enhanced certificate transparency

Mark D. Ryan
School of Computer Science
University of Birmingham




Network and Distributed System Security Symposium
25 February 2014

# Email today

- The age of "electronic mail" may soon be upon us. . . (1978)
- Attackers:
  - Governments and security agencies
  - Corporations whose business model is to monitise our data

# End-to-end encrypted mail

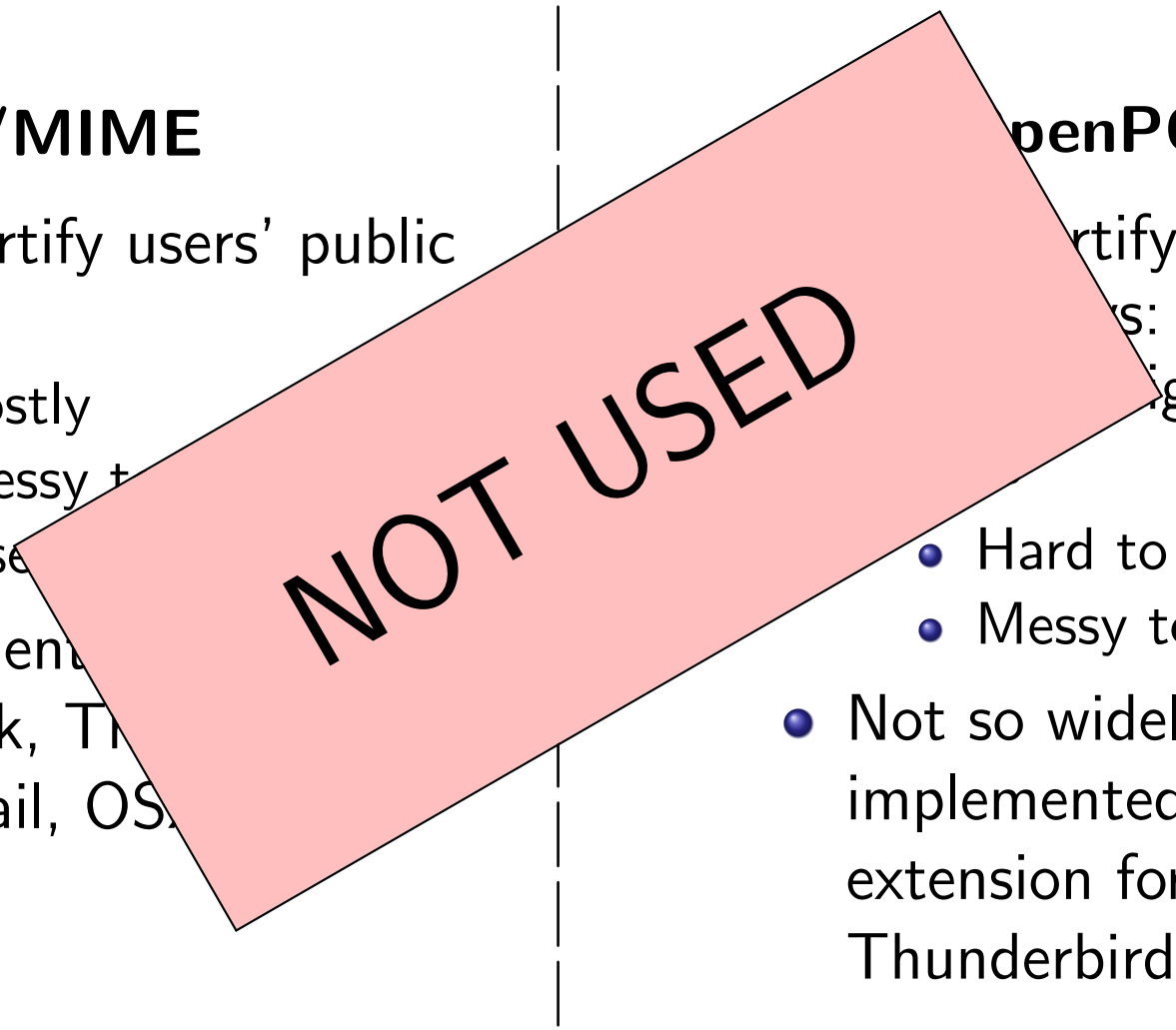
## S/MIME

- CAs certify users' public keys
  - Costly
  - Messy to set up
  - Insecure
- Implemented in Outlook, Thunderbird, iOS Mail, OSX,...

## OpenPGP

- 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)

# End-to-end encrypted mail



**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

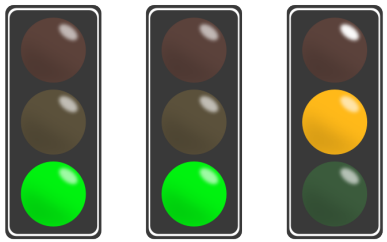
# CT-Mail

To: Joe Bloggs <joe.bloggs@example.com> ✔

Cc: Alice Smith <alice@alice-n-bob.com> ✗

Subject: Meeting tonight

Healthiness checks



Hi Joe, Bob's away on business.

# Certificate transparency [Laurie, Kasper, Langley 2012]

**Aim:** ensure that whenever a CA signs a certificate, there is persistent evidence of this fact. A CA cannot sign certificates inadvertently/sneakily.

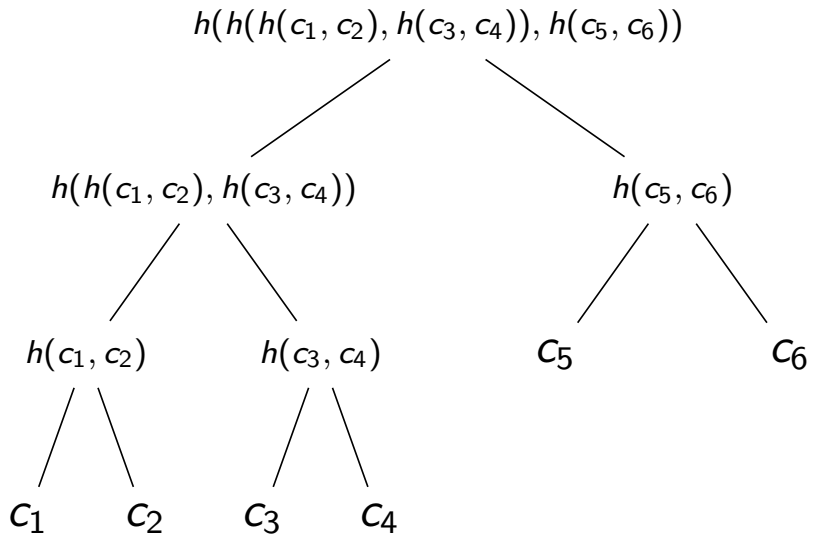
**Mechanism:** a certificate is accepted only if it is included in the *append-only public log* of certificates issued by the given CA.

The certificate comes with proof that it is included in the log.

Users' client software checks that log is append-only and linear.

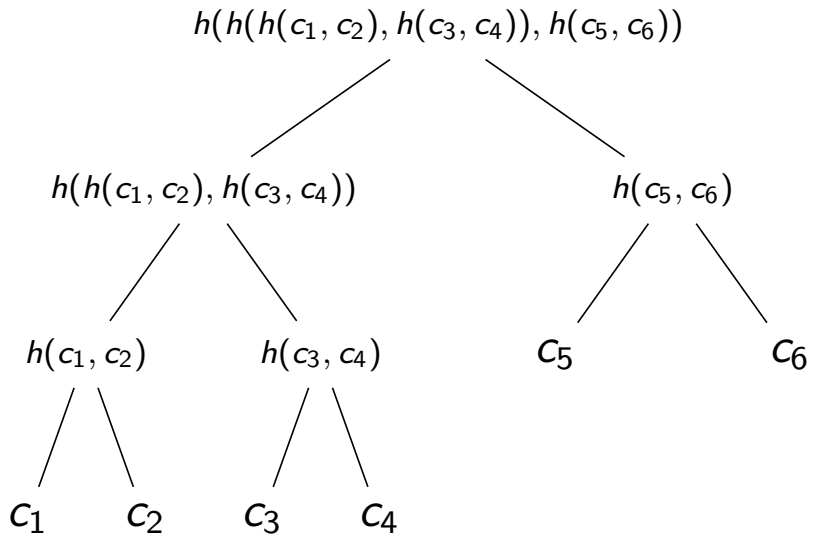
**Status:** IETF draft; RFC; being implemented in Chrome.

# Certificate transparency: append-only public log




| Algorithm | Complexity | Typical size $10^9$ certif. |
|---|---|---|
| request_h() | $O(1)$ | |
| 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



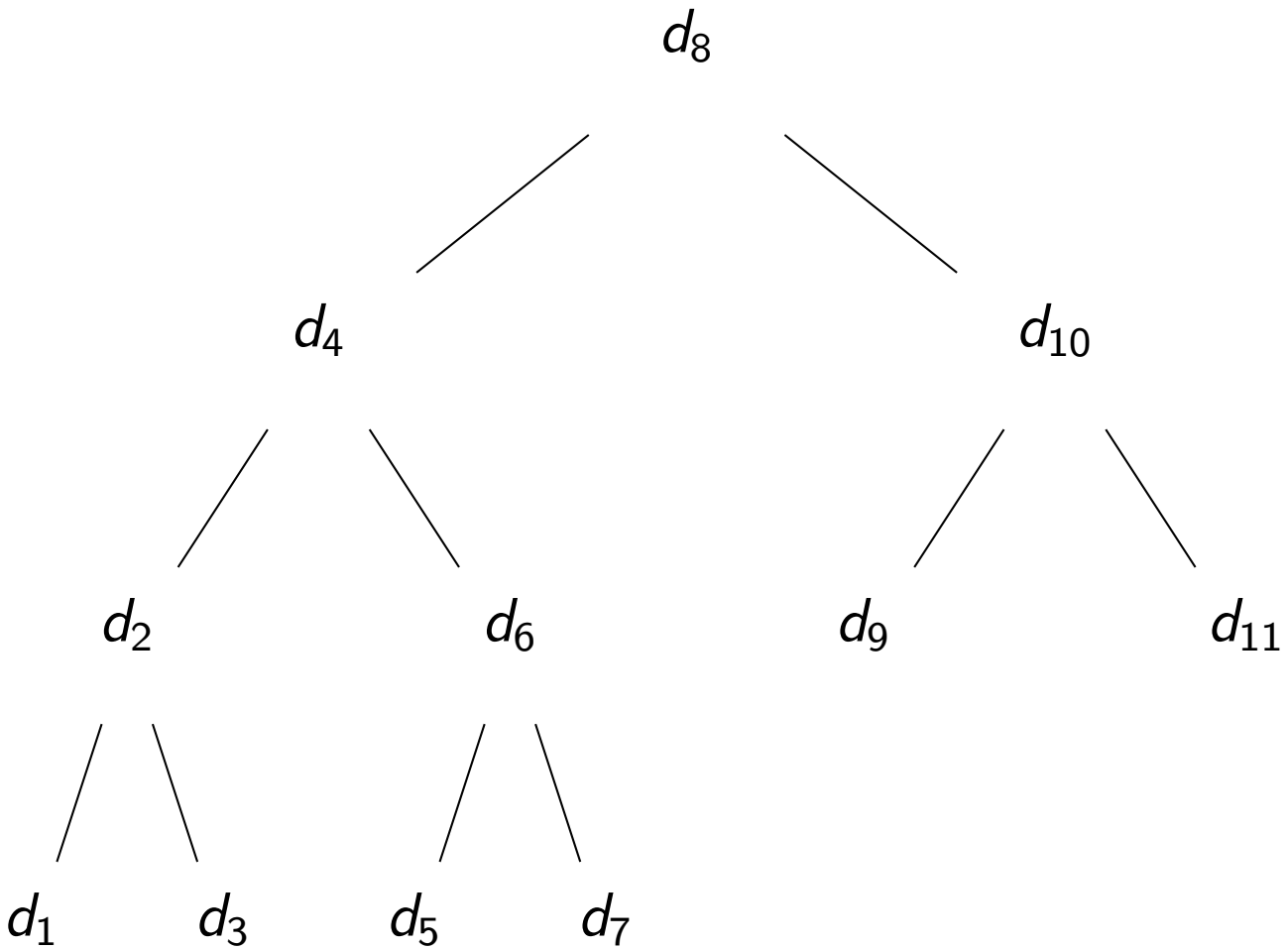

| Algorithm | Complexity | Typical size $10^9$ certif. |
|---|---|---|
| request_h() | $O(1)$ | 0.25 KB |
| prove_presence($h$, *cert*) | $O(\log n)$ | 2 KB |
| prove_absence($h$, *cert*) | $O(n)$ | 60 GB |
| prove_extension($h_1$, $h_2$) | $O(\log n)$ | 2 KB |

# Key revocation

- Cert. transp. doesn't support proofs of absence
  - Therefore it does not support key revocation:
    current $=$ present $\wedge$ $\neg$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

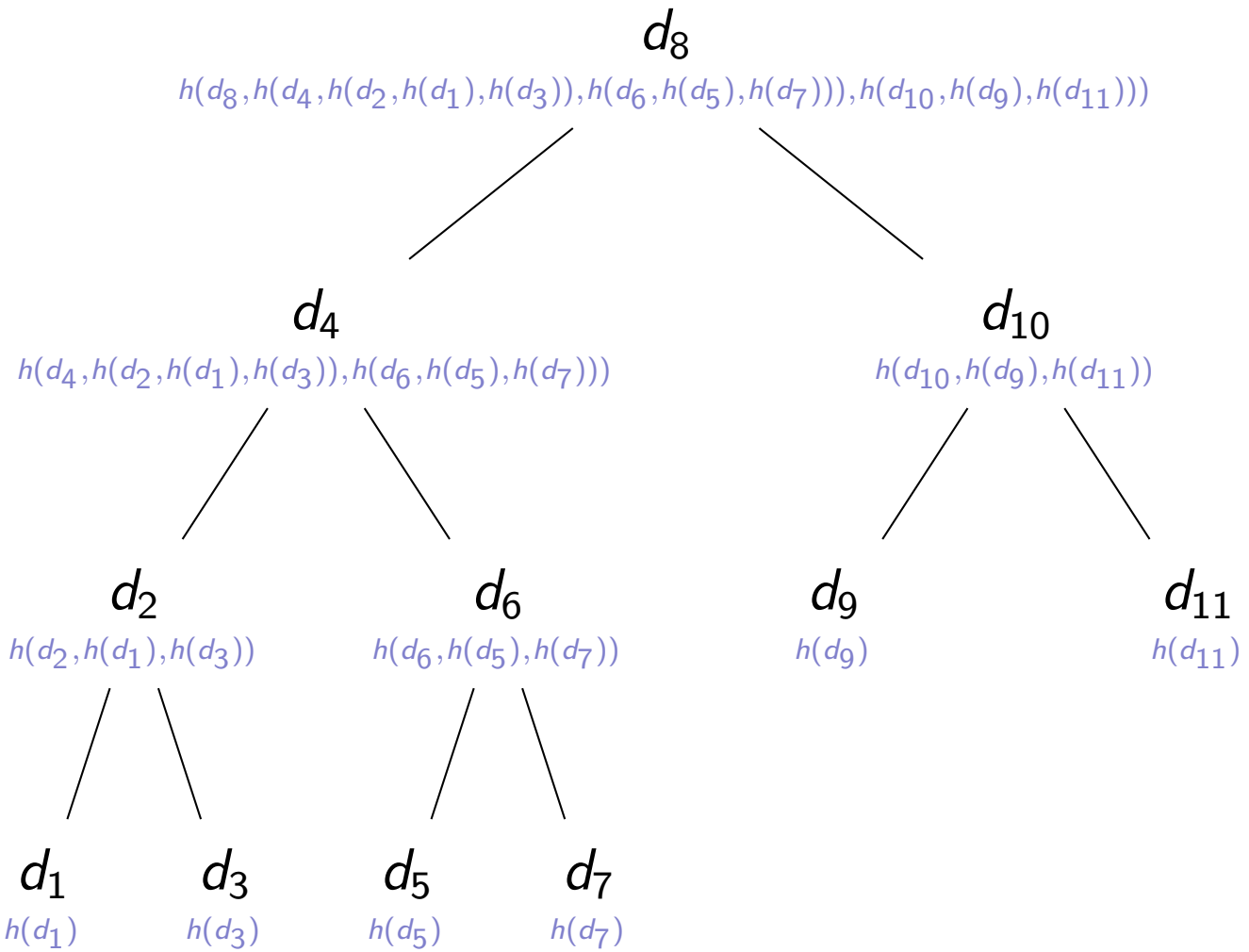
# Proofs of currency or absence

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



# Proofs of currency or absence

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



# Proof of absence

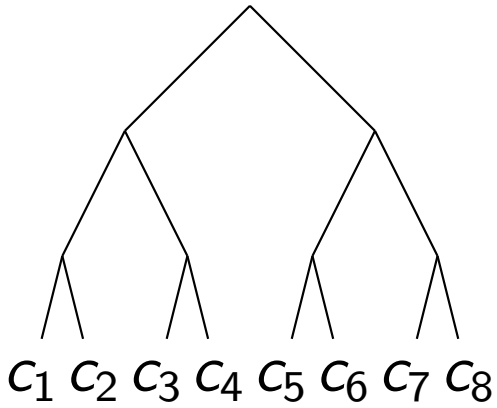
To prove there is no key for $subj$, the log maintainer provides:

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

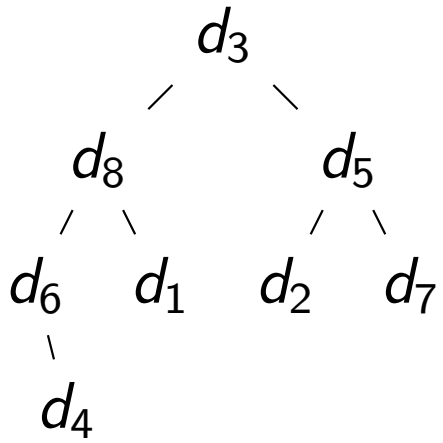
Client verifies the proofs, and also that
$subj_1 < subj < subj_2$ lexicographically.

# Certificate Issuance and Revocation Transparency

**ChronTree**




**LexTree**

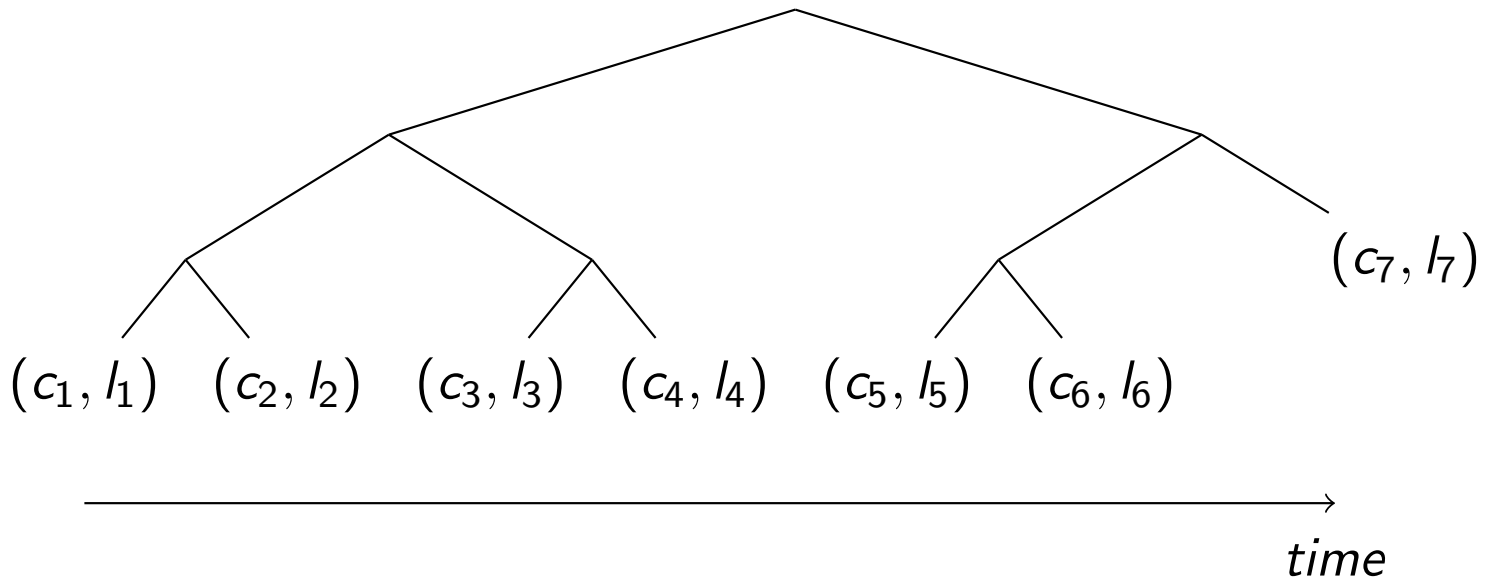



| **Proof of** | | |
|---|---|---|
| presence | $O(log\, n)$ | $O(log\, n)$ |
| absence | $O(n)$ | $O(log\, n)$ |
| extension | $O(log\, n)$ | $O(n)$ |
| consistency | $O(n)$ | |

# Consistency checking

Two ways to check ChronTree/LexTree sync:

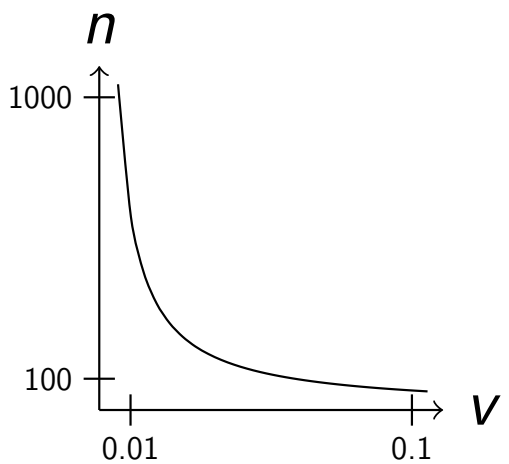
- Total: receive all updates, and check everything.
- Random: user client software specifies random $(c_i, l_i)$, and requests proof that $\mathrm{LT}(l_i) = \mathrm{LT}(l_{i-1}) + c_i$.



# Coverage of random checking

- $n$ number of users
- $v$ proportion of 'victims' (CA is cheating about their certificates)
- $t$ time in days until detection with probability 0.5

$t = 0.1$ 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.

# CT-Mail

**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 $pk_{Bob}$.

| *Realities of email* | *How handled* |
|---|---|
| Multiple devices | Store keys in $\{\text{keypurse}\}_k$ in cloud<br>Enroll new device by transferring $k$<br>Verify $h_s \sqsubseteq h \sqsubseteq h'$ |
| Plaintext compat. | UI informs of encr. status |
| Webmail | OSS browser extension |
| Search | Restrict it to headers<br>Optionally, store $\text{HMAC}_k(\text{word})$ |
| Metadata prot'n, OTR | Not realities |

| *Realities of email* | *Remark* |
|---|---|
| Password forgotten | Usual methods |
| Password compromised | Usual methods |
| $k$ "forgotten" | Lose store; reset account |
| $k$ compromised | Past email may be compr.<br>Revoke pk; reset account |

# Why do you want end-to-end encrypted mail?

**Drugs, guns, paedophilia**

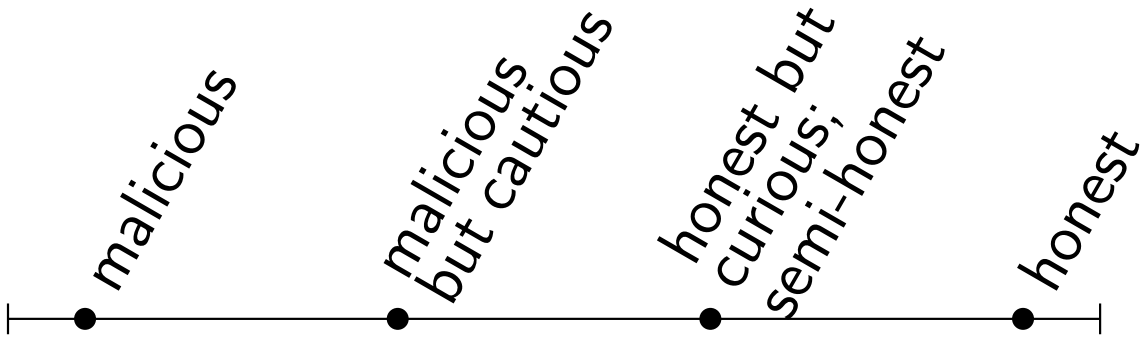
- You need to prevent attacks, not just detect them
- You should consider your provider to be malicious
- **CT-Mail can't help you**

**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)**

# Attacker models



# Summary

- Certificate transparency
- Certificate issuance and revocation transparency (CIRT)
- CT-Mail
  - Usability.
- Malicious-but-cautious attacker
  - Applications
  - Formalisation
  - Analysis/verification