Revisiting User Privacy for Certificate Transparency

Daniel Kales, Olamide Omolola, Sebastian Ramacher

EuroS&P’19, June 19th, 2019

> https://www.iaik.tugraz.at
Revisiting User Privacy for Certificate Transparency

Outline

1. What is Certificate Transparency?
2. Privacy Concerns for End Users
3. Solving Privacy Concerns
4. Practical Evaluation
What is Certificate Transparency?
What is Certificate Transparency?

X509 Certificate Ecosystem

- Certificates bind user’s information to the public key
- Certificate is signed by a root certificate
- Root certificate is owned by a trusted entity called Certificate Authority (CA)
- User’s certificate can be verified by linking it to the known root certificate
What is Certificate Transparency?

**X509 Certificate Ecosystem Problems**

- Signatures prevent malicious websites from using forged certificates.
- No protection against mistakenly or maliciously issued certificates!
- Real-world problems: DigiNotar compromised by hackers.
What is Certificate Transparency?

Certificate Transparency

- Need monitoring system for all issued certificates
- Goals:
  - easily accessible to everyone, open framework
  - refuse use of certificates not in monitoring system
  - cryptographic guarantees for logging
- **Certificate Transparency** [Lau14; LLK13] was designed to be this system
- Log servers give signed promise of inclusion in log to CA
  - Signed Certificate Timestamp (SCT)
- Mandatory for certificates issued after April 30th, 2018!
What is Certificate Transparency?

Certificate Transparency (cont.)

Client, Auditor  \[\xrightarrow{\text{TLS handshake}}\] Web Server  \[\xrightarrow{\text{issue cert. with SCT}}\] Cert. Authority  \[\xrightarrow{\text{SCT}}\] Log Server  \[\xrightarrow{\text{add cert. to log}}\] Cert. Authority  \[\xrightarrow{\text{Consistency checks}}\] Monitor  \[\xrightarrow{\text{check if cert. is logged}}\] Log Server  \[\xrightarrow{\text{check if cert. is logged}}\] Client, Auditor
What is Certificate Transparency?

Log Server Structure

- Merkle tree
  - Binary tree of hashed nodes
  - Log server periodically updates tree with new certificates
  - Log server also signs root hash

Diagram:

- Root node `r`
- Child nodes: `m`, `n`
- Child nodes of `m`: `i`, `j`
- Child nodes of `n`: `k`, `l`
- Child nodes of `i`: `a`, `b`, `c`
- Child nodes of `j`: `d`, `e`, `f`
- Child nodes of `k`: `g`, `h`
- Child nodes of `l`: None

Hashes:
- `a`: `c0`
- `b`: `c1`
- `c`: `c2`
- `d`: `c3`
- `e`: `c4`
- `f`: `c5`
- `g`: `c6`
- `h`: `c7`
What is Certificate Transparency?

Log Server Structure (cont.)

- Membership proof
  - Release intermediary hashes
  - Re-calculate path to root
  - Compare against known root hash
  - Logarithmic proof size
Privacy Concerns for End Users
Privacy Concerns for End Users

- End users have auditing role
  - Verify certificate is contained in log according to SCT
  - If not, report log server as malicious

- Privacy loss:
  - Log server learns browsing behavior of client
  - Could deter clients from using Certificate Transparency

Alice

is the cert. for bob.site in the log?

Log Server

Alice visits bob.site, how interesting!
Solving Privacy Concerns
Naïve Solutions

- Download full log:
  - Infeasible for most clients
  - Log sizes of 10+ GiB

- Redirect query through proxy:
  - Protect client query from log server
  - Only shifts privacy problem to proxy
Solving Privacy Concerns

Other Approaches

- **Stapling Approach:**
  - Web server gets proof from log server
  - Forwards proof to client
  - More work for web server

- **CT over DNS:**
  - Get proofs via DNS queries
  - Shifts privacy concerns to DNS server
  - DNS mostly plaintext
Private Information Retrieval

- Retrieve item from database

Alice

\[ i \]

Item \( i \)
Private Information Retrieval

- Retrieve item from database
- Without revealing accessed item
Private Information Retrieval (cont.)

- Previous efforts by Lueks and Goldberg [LG15] in 2015
  - Optimizations to Percy++ PIR system
  - Multi-server model
  - Speedup when answering many client queries at once
  - Use-case: Certificate Transparency
- Assumed 4 million certificates
  - Runtime of a few seconds per query
  - Practical today?
Solving Privacy Concerns

Current Log Server Statistics

- **merkle.town**: CT ecosystem statistics

- Number of new certificates per hour (global): \( \approx 53,000 \)

<table>
<thead>
<tr>
<th>Root CA</th>
<th>Certificates</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>DigiCert</td>
<td>64,226,041</td>
<td>( 2^{25.94} )</td>
</tr>
<tr>
<td>Let’s Encrypt</td>
<td>941,016,262</td>
<td>( 2^{29.81} )</td>
</tr>
<tr>
<td>Sectigo</td>
<td>246,484,842</td>
<td>( 2^{27.88} )</td>
</tr>
<tr>
<td>Other</td>
<td>62,114,615</td>
<td>( 2^{25.89} )</td>
</tr>
</tbody>
</table>

\(^1\)retrieved on 2019-04-08
Solving Privacy Concerns

Changes to Merkle Tree Structure

- Original Merkle Tree:
Solving Privacy Concerns

Changes to Merkle Tree Structure (cont.)

- Merkle Tree with Sub-Trees:

```
  r1
   i
    a
c0
   j
    b
    c
c1
e
   k
    f
c5
d
   l
    g
c6
  r2
    h
c7
  ...
```

```
r
  u
  r1
  v
  r2
  r3
  r4
```
Changes to Merkle Tree Structure (cont.)

- Log server only **issues** SCTs **once per** predetermined **time span**
  - e.g., a new sub-tree every hour (time span configurable)
- By completing full sub-tree, we can **include the proof** in the SCT
  - store Merkle tree proofs for sub-trees in SCT **extension** field
  - only retrieve proof between sub-tree root and top-level root hashes
- Tradeoff between tree size, SCT issuing latency and SCT size
- Other accumulators possible (e.g., bilinear accumulators)
Solving Privacy Concerns

Multi-Server PIR

- Multi-Server PIR gives information-theoretic security
  - Even with unlimited computing power, no way for server to find index $i$!
- Important restriction: **No collusion** between servers!
- Much better performance than single-server PIR
  - No need for expensive primitives, e.g. homomorphic encryption
  - Based on secret-sharing approaches
Solving Privacy Concerns

Multi-Server PIR (cont.)

- “Linear-Summation Scheme” [CGK’95]

![Diagram of multi-server PIR]

- Client
- DB 1
- DB 2
Solving Privacy Concerns

Multi-Server PIR (cont.)

- “Linear-Summation Scheme” [CGK'95]

\[ r_1 = x_1 \oplus x_3 \oplus x_4 \]

\[ r_2 = x_1 \oplus x_2 \oplus x_3 \oplus x_4 \]

\[ x_2 = r_1 \oplus r_2 \]
Solving Privacy Concerns

Two-Server PIR from DPFs

- Problem: Still $N = |DB|$ bits of communication per server and query
- Distributed Point Functions (DPF) [GI14]
  - “Function Secret Sharing” by Boyle et al. [BGI15]
  - $(k_1, k_2) \leftarrow \text{DPF.Gen}(N, q)$
    - Generate two short $(\log N)$ keys based on chosen index $q$ and length $N$
  - $K_i \leftarrow \text{DPF.Eval}(N, k_i)$
    - Expand short key $k_i$ to $N$ bit long keystream $K_i$
- Property: $K_1 \oplus K_2$ is a bitstring with only one bit at position $q$ set
Solving Privacy Concerns

Two-Server PIR from DPFs (cont.)

\[(k_1, k_2) = \text{DPF.Gen}(6, 2)\]

\[010110 = \text{DPF.Eval}(6, k_1)\]

\[011110 = \text{DPF.Eval}(6, k_2)\]
Solving Privacy Concerns

Multi-Server PIR Deployment

- Important requirement: **No collusion between two servers!**
  - If violated, privacy is lost!

- Real-world deployment:
  - Log server data is publicly accessible
  - Competitor of first log server: Google ↔ Microsoft
  - Privacy-conscious organizations: EFF, EDRi

- Only extension to normal log server API, users still can query without privacy protection
Practical Evaluation
Evaluation

- Single-Server PIR
  - Not feasible for full CT logs with \(2^{28}\) or more elements!
  - Open-source PIR framework XPIR [MBF+16]
  - Evaluation for sub-tree each hour

<table>
<thead>
<tr>
<th>tree size</th>
<th>DB gen. [ms]</th>
<th>Query gen. [ms]</th>
<th>Reply gen.</th>
<th>Comm. [KB]</th>
</tr>
</thead>
<tbody>
<tr>
<td>(2^{15})</td>
<td>3640</td>
<td>7491</td>
<td>1748</td>
<td>128</td>
</tr>
</tbody>
</table>
Evaluation (cont.)

- Multi-Server PIR using DPFs
- Almost feasible even for full CT logs!

<table>
<thead>
<tr>
<th>tree size</th>
<th>DPF.Gen [ms]</th>
<th>DPF.Eval [ms]</th>
<th>XOR</th>
<th>Total [ms]</th>
<th>Comm. [B]</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2^{20}$</td>
<td>0.05</td>
<td>0.32</td>
<td>4.28</td>
<td>4.66</td>
<td>2938</td>
</tr>
<tr>
<td>$2^{22}$</td>
<td>0.07</td>
<td>1.23</td>
<td>16.72</td>
<td>18.03</td>
<td>3590</td>
</tr>
<tr>
<td>$2^{24}$</td>
<td>0.08</td>
<td>4.78</td>
<td>64.49</td>
<td>69.36</td>
<td>4314</td>
</tr>
<tr>
<td>$2^{26}$</td>
<td>0.09</td>
<td>19.22</td>
<td>251.32</td>
<td>270.64</td>
<td>5110</td>
</tr>
<tr>
<td>$2^{28}$</td>
<td>0.11</td>
<td>78.41</td>
<td>988.93</td>
<td>1067.46</td>
<td>5978</td>
</tr>
</tbody>
</table>
Evaluation (cont.)

- Multi-Server PIR using DPFs and sub-accumulators
  - Overhead less than 10 ms and 4 KB for full CT log

<table>
<thead>
<tr>
<th>$N$</th>
<th>$N_\Lambda$</th>
<th>Sub-acc. type</th>
<th>$N_{sub}$</th>
<th>DPF total [ms]</th>
<th>Acc. verify [B]</th>
<th>Com. extra [B]</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2^{31}$</td>
<td>$2^{15}$</td>
<td>RSA</td>
<td>$2^{16}$</td>
<td>3.97</td>
<td></td>
<td>384</td>
</tr>
<tr>
<td>$2^{31}$</td>
<td>$2^{15}$</td>
<td>Bilinear</td>
<td>$2^{16}$</td>
<td>0.13</td>
<td>$&lt; 0.01$</td>
<td>1623</td>
</tr>
<tr>
<td>$2^{31}$</td>
<td>$2^{15}$</td>
<td>Merkle</td>
<td>$2^{16}$</td>
<td>2.81</td>
<td></td>
<td>512</td>
</tr>
<tr>
<td>$2^{31}$</td>
<td>$2^{21}$</td>
<td>RSA</td>
<td>$2^{10}$</td>
<td>3.97</td>
<td></td>
<td>384</td>
</tr>
<tr>
<td>$2^{31}$</td>
<td>$2^{21}$</td>
<td>Bilinear</td>
<td>$2^{10}$</td>
<td>8.36</td>
<td>$&lt; 0.01$</td>
<td>3255</td>
</tr>
<tr>
<td>$2^{31}$</td>
<td>$2^{21}$</td>
<td>Merkle</td>
<td>$2^{10}$</td>
<td>2.81</td>
<td></td>
<td>320</td>
</tr>
</tbody>
</table>
Practical Evaluation

Conclusion

- Changes to Merkle-Tree structure enable less costly PIR queries
- Sub-tree structure generalizes to other types of accumulators
- Multi-server PIR based on DPFs with sub-accumulators
  - Overhead less than 10 ms and 4 KB for full CT log
- Multi-server PIR possible without major changes in CT ecosystem
- Optional for users if they want privacy, compatible with old API
Questions?

Implementation:

- DPF (in Go): [https://github.com/dkales/dpf-go](https://github.com/dkales/dpf-go)
- Log server: [https://github.com/dkales/certificate-transparency](https://github.com/dkales/certificate-transparency)
Practical Evaluation

References I


