Private Blocklist Lookups with Checklist

Dima Kogan
Stanford

Henry Corrigan-Gibbs
MIT

USENIX Security Symposium 2021
Blocklist lookup

Server holds a blocklist of strings

Client holds a **private** string it wants to check against the blocklist

Examples:
- Certificate revocation
- Password checkup
- Safe Browsing

Blocklist \( \mathcal{B} \subseteq \{0,1\}^* \)
Safe Browsing

Firefox browser

Partially hashes
- 0x104
- 0x130
- 0x1F3
- 0x1FF
- 0x24C
- 0x2B2

Warn

Firefox browser

Leaks information about user’s browsing history [GKL16,Per19,BK20]

Google Safe Browsing

Lookup

Partial hashes
- 0x24C

Full hash
- 0x24C1A8

Hashes of ~3M dangerous URLs
- 0x104A158413B9...
- 0x130CA8F45BE3...
- 0x1F346FFD10DA...
- 0x1FF60910E4BF1...
- 0x24CC8335003A...
- 0x2B26ED8D140B...

Leaves information about user's browsing history [GKL16,Per19,BK20]
This work

**Checklist** – a system for private blocklist lookups

Builds on **offline/online private information retrieval** [CK20]

- Allows for **sublinear** online server time

**Contributions:**

- New offline/online PIR (reduces server computation by >100x)
- A technique to efficiently support database updates
- Implementation & evaluation of a private Safe Browsing system
Requirements

Correctness
Client learns whether $s \in B$ (with overwhelming prob.)

Privacy for the client
(Malicious) server “learns nothing” about client’s string

Non-goal: privacy for the server

(see paper for discussion of this extension)

Efficiency
Minimize latency, communication, computation, storage
Two-server private information retrieval
[CGKS95,…]

Server computation is large
Servers needs to do $\Omega(n)$ work [BIM04]

Communication is small
$O(\log n)$ from DPFs [GI14, BGI15]
(Distributed Point Functions)

Security
Each server **on its own** learns nothing about $i$

$x \in \{0,1\}^n$

Index
$i \in [n]$

$[n] = \{1, \ldots, n\}$

$x_i \in \{0,1\}$
**Offline/Online PIR [CK20]**

**Step 1: Offline phase**

The left server runs in linear time.

But only once per client.

$\lambda \approx \sqrt{n}$ bits

$\lambda -$ security parameter ($\approx 128$)
Offline/Online PIR [CK20]

Step 2: Online phase – reading $x_i$

Client can repeat online phase to read multiple items

Client time is large

Index $i \in [n]$

Hint

Sublinear online time

Client can repeat online phase to read multiple items

Client time is large
Our first contribution: \( \lambda \times \) faster offline/online PIR

\[ x \in \{0,1\}^n \]

\( o(n) \) time

Improve server time from \( \lambda \sqrt{n} \) to \( \sqrt{n} \)

\( o(n) \) time

Improve communication from \( \lambda^2 \log n \) to \( \lambda \log n \)

\( x_i \in \{0,1\} \)

\( \lambda \) – security parameter (\( \approx 128 \))
Our second contribution: Offline-online PIR with DB updates

When the database changes, client needs a new hint

Naïve approach: rerun the offline phase after each change
• **Linear** amount of server work on each change

Refined approach: incremental preprocessing
• Use “buckets” of exponentially increasing size
• **Logarithmic** amount of server work on each change

Static-to-dynamic data structures \([BS80,GO96,CKO90,SSP13,SPS14,PT16]\)
Our third contribution: Private Safe Browsing

Checklist: a system for private Safe Browsing queries, integrated and evaluated with Firefox browser
Implementation

Roughly 2500 lines of Go + 500 lines of C

Support offline/online and DPF-based PIR

Browser integration: local proxy + browser config change
Estimating the Safe Browsing parameters

Monitor a week of Safe Browsing requests and responses

Local proxy forwards requests to the real Safe Browsing service

Deduce:
- Frequency of lookups
- Frequency of updates
- Database growth
Checklist with offline/online PIR is more efficient for the server

Replay recorded trace of a single user

Server running on e2-standard-4 Google Compute Engine machine (4 vCPUs, 16 GB RAM)
Checklist with DPF-based PIR is more efficient for the client

Client running on a Pixel 5 mobile phone
# Overview of results

<table>
<thead>
<tr>
<th>Approach</th>
<th>Server Costs (servers per 1B users)</th>
<th>Latency (ms)</th>
<th>Client computation</th>
<th>Communication</th>
<th>Client Storage</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Initial (sec)</td>
<td>Running (sec/month)</td>
<td>Initial (MB)</td>
<td>Running (MB/month)</td>
<td>Initial (MB)</td>
</tr>
</tbody>
</table>
## Overview of results

<table>
<thead>
<tr>
<th>Approach</th>
<th>Server Costs (servers per 1B users)</th>
<th>Latency (ms)</th>
<th>Client computation</th>
<th>Communication</th>
<th>Client Storage</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Initial (sec)</td>
<td>Initial (MB)</td>
<td>Initial (MB)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Running (sec/month)</td>
<td>Running (MB/month)</td>
<td>Running (MB/month)</td>
</tr>
<tr>
<td>Non-private</td>
<td>143</td>
<td>91</td>
<td>3.1</td>
<td>5.0</td>
<td>4.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.5</td>
<td>3.0</td>
<td>0.2</td>
</tr>
</tbody>
</table>
# Overview of results

<table>
<thead>
<tr>
<th>Approach</th>
<th>Server Costs (servers per 1B users)</th>
<th>Latency (ms)</th>
<th>Client computation Initial (sec)</th>
<th>Running (sec/month)</th>
<th>Communication Initial (MB)</th>
<th>Running (MB/month)</th>
<th>Client Storage Initial (MB)</th>
<th>Running (MB/month)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-private</td>
<td>143</td>
<td>91</td>
<td>3.1</td>
<td>0.5</td>
<td>5.0</td>
<td>3.0</td>
<td>4.3</td>
<td>0.2</td>
</tr>
<tr>
<td>Full list</td>
<td>Very small – not measured</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Overview of results

<table>
<thead>
<tr>
<th>Approach</th>
<th>Server Costs (servers per 1B users)</th>
<th>Latency (ms)</th>
<th>Client computation</th>
<th>Communication</th>
<th>Client Storage</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Initial (sec)</td>
<td>Running (sec/month)</td>
<td>Initial (MB)</td>
<td>Running (MB/month)</td>
<td>Initial (MB)</td>
</tr>
<tr>
<td>Non-private</td>
<td>143</td>
<td>91</td>
<td>3.1</td>
<td>5.0</td>
<td>4.3</td>
</tr>
<tr>
<td>Full list</td>
<td>Very small – not measured</td>
<td>91.8</td>
<td>13.2</td>
<td>91.8</td>
<td>4.5</td>
</tr>
<tr>
<td>DPF PIR</td>
<td>9047</td>
<td>122</td>
<td>2.6</td>
<td>5.0</td>
<td>4.3</td>
</tr>
</tbody>
</table>
# Overview of results

<table>
<thead>
<tr>
<th>Approach</th>
<th>Server Costs (servers per 1B users)</th>
<th>Latency (ms)</th>
<th>Client computation</th>
<th>Communication</th>
<th>Client Storage</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Initial (sec)</td>
<td>Running (sec/month)</td>
<td>Initial (MB)</td>
</tr>
<tr>
<td>Non-private</td>
<td>143</td>
<td>91</td>
<td>3.1</td>
<td>0.5</td>
<td>5.0</td>
</tr>
<tr>
<td>Full list</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DPF PIR</td>
<td>9047</td>
<td>122</td>
<td>2.6</td>
<td>0.8</td>
<td>5.0</td>
</tr>
<tr>
<td>Offline/online PIR</td>
<td>1348</td>
<td>90</td>
<td>13.3</td>
<td>8.0</td>
<td>10.3</td>
</tr>
</tbody>
</table>

Note: Very small – not measured
Conclusion & future work

Two-server PIR is a practical tool for privacy preserving systems
• Several alternatives that allow for different trade-offs

Future direction: better single-server offline/online PIR
Conclusion & future work

Two-server PIR is a practical tool for privacy preserving systems
• Several alternatives that allow for different trade-offs

Future direction: better single-server offline/online PIR

dkogan@cs.stanford.edu  henrycg@csail.mit.edu
https://eprint.iacr.org/2021/345
https://github.com/dimakogan/checklist