Security and Privacy of Hash-Based Software Applications

This work has been partially supported by the LabEx PERSYVAL-Lab (ANR-11-LABX-0025-01) funded by the French program Investissement d’avenir.

Amrit Kumar

October 20, 2016

Privatics team, Inria
Université Grenoble Alpes
Hashing

- A function $h : \{0, 1\}^* \rightarrow \{0, 1\}^\ell$, where $\ell$ is the digest size.
- Cryptographic: (second) pre-image and collision resistant.

**Pre-image resistance**

```
?  \downarrow
\hline
h  \downarrow
h(x)
```

**2nd pre-image resistance**

```
?  \downarrow
\hline
x \neq ?
 \downarrow
h  \downarrow
h  \downarrow
h(x) = h(x')
```

**Collision resistance**

```
?  \downarrow
\hline
? \neq ?
 \downarrow
h  \downarrow
h  \downarrow
h(x) = h(x')
```

**Best generic attack**

- Pre-image resistance: $2^\ell$
- 2nd pre-image resistance: $2^\ell$
- Collision resistance: $2^{\ell/2}$
Are collisions always bad? (I)

A simple use case:

- Instead of storing \( n \) (large) data items, store their digests.
- If \( \ell \) is large, collisions are hard to find \( \Rightarrow \) space required \( = n \times \ell \) bits.

Collisions for further space savings:

- \( d_i \) now substitutes both \( x_1 \) and \( x_2 \) \( \Rightarrow \) space required \( < n \times \ell \) bits.
- **Caveat**: May introduce some unexpected behavior.
- Core of several efficient (probabilistic) data structures:
  - Bloom filters for membership testing
  - Sketches for data stream analysis
Are collisions always bad? (II)

Use case in privacy:

- Hashing as a pseudonymization technique.
- If $\ell$ is large, but the number of identifiers $n$ is enumerable (in reasonable time)
  - Exhaustive search breaks pseudonymization.

\[
\ell \text{ is large} \quad \ell \text{ is small}
\]

\[
\begin{align*}
    d_1 & \leftarrow h \\
    h & \quad h \\
    d_2 & \leftarrow h \n\end{align*}
\]

- If $\ell$ is sufficiently small:
  - On average $n/2^\ell$ identifiers share the same pseudonym.
  - Notion of anonymity-set.
  - **Caveat**: Provides weak anonymity guarantees.
  - Employed in Google Safe Browsing: a malicious URL detection tool.
Contrasting perspectives

- Collisions have to be absolutely avoided in cryptography.
- Somewhat welcome in algorithms and data structures.
- Useful to some extent in the context of privacy.

**Goal:** Investigate the security and privacy implications of hash collisions.

**Focus for today:**

- **Security:** Bloom Filters
  1. *The Power of Evil Choices in Bloom Filters.* DSN’15
     Joint work with T. Gerbet and C. Lauradoux
  2. *Bloom Filters in Adversarial Settings.* Under submission
     Joint work with C. Lauradoux and P. Lafourcade

- **Privacy:** Safe Browsing
  1. *A Privacy Analysis of Google and Yandex Safe Browsing.* DSN’16
     Joint work with T. Gerbet and C. Lauradoux
Security: Bloom Filters
Bloom filters [Bloom 1970]

Setup\((m, n, k)\):

- A binary vector \(\mathbf{z}\) of size \(m\) compressing a set of \(n\) items.
- \(k\) uniform and independent hash functions: \(h_i : \{0, 1\}^* \rightarrow [0, m - 1]\)
- \(\mathbf{z}\) initialized to \(\mathbf{0}\).

Operations:

- **Insert**\((x)\): Set bits of \(\mathbf{z}\) at \(h_1(x), \ldots, h_k(x)\) to 1.

\[
S = \{x_1, x_2, x_3\} \quad k = 2
\]

\[
\begin{array}{cccccccc}
0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1
\end{array}
\]

- **Query**\((y)\): Return True if bits of \(\mathbf{z}\) at \(h_1(y), \ldots, h_k(y)\) are all 1.
- False positive rate and its optimum value have been well studied.
Our contributions

- **Define adversary models** for Bloom filters.
  - Query-only adversary
  - Chosen-insertion adversary
  - Deletion adversary
    - Specific to counting Bloom filters (not covered today)

- **DoS attacks** on Bloom enabled software applications:
  - Increase false positive probability,
  - Increase query time.

- **Worst-case analysis of Bloom filters:**
  - false-positive probability,
  - new filter parameters.

- Bloom hash tables as a **potential replacement for Bloom filters**.
Query-only adversary

**Capabilities:** Only queries to the filter.

**Assumption:** State of the filter is known.

**Goals:**

- Craft items that generate false positives
  - Probability to forge a false positive is: \( \left( \frac{w_H(\vec{z})}{m} \right)^k \)
  - \( w_H(\cdot) \) is the Hamming weight.

- Or, items whose processing leads to latency.
  - First \( k - 1 \) bits are set to 1 and the \( k \)-th bit set to 0.

\[
y = \begin{cases} 1 & y = 3 \\ \text{0 otherwise} \end{cases}
\]

\[
\begin{array}{cccccccccccc}
1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\
\end{array}
\]

- The probability of finding such an item is:

\[
\frac{(m - w_H(\vec{z})) \cdot \binom{w_H(\vec{z})}{k-1}}{m^k}.
\]
Chosen-insertion adversary

Capabilities: Can choose items to insert in the filter.
Assumption: State of the filter is known.
Goal: Increase the false positive probability.

Strategy: Greedily insert $x$ that maximizes #bits set to 1.

- Each $x$ sets $k$ bits to 1.

Impact:

<table>
<thead>
<tr>
<th></th>
<th>No attack</th>
<th>Under attack</th>
</tr>
</thead>
<tbody>
<tr>
<td>#bits set to 1</td>
<td>$0.72nk_{opt}$</td>
<td>$nk_{opt}$</td>
</tr>
<tr>
<td>false positive rate ($f$)</td>
<td>$\frac{1}{2} k_{opt}$</td>
<td>$\left(\frac{nk_{opt}}{m}\right)^{k_{opt}}$</td>
</tr>
</tbody>
</table>
Impact on a sample filter

**Parameters:** $m = 3200$, $n = 600$, $k_{opt} = 4$, $f_{opt} = 0.077$

![Graph showing false positive probability versus number of inserted items.](image-url)
Applying adversary models

Factors enabling our attacks:

- Insecure hash functions.
- Digest truncation.
- High Bloom filter false positive rate.

Vulnerable software applications:

<table>
<thead>
<tr>
<th>Software app.</th>
<th>Hashing</th>
<th>Parameter info.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scrapy: Web crawler</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>Dablooms: Spam filter</td>
<td>MurmurHash</td>
<td>$n = 100000, f = 0.057$</td>
</tr>
<tr>
<td>Squid: Web proxy</td>
<td>MD5</td>
<td>$f = 0.09, k = 4$</td>
</tr>
<tr>
<td>AIEngine: NIDS</td>
<td>C++ hash</td>
<td>$\ell = 13, n = 5000, f = 0.45$</td>
</tr>
<tr>
<td>NSRL: Forensic tool</td>
<td>SHA-1</td>
<td>$\ell = 32, n \approx 14 \times 10^6, f = 8.08 \times 10^{-10}$</td>
</tr>
<tr>
<td>sdhash: Forensic tool</td>
<td>SHA-1</td>
<td>$\ell = 11, n = 128, f = 0.0014$</td>
</tr>
</tbody>
</table>
NSRL forensic tool:

- A whitelist of “known safe files”.
- Stored and distributed as a Bloom filter.
- Maintained by NIST.

A query-only attack: Goal is to hide a contraband file.

- Adversary modifies the file to create a false positive.
- Modification should be easily reversible.
- The filter detects the file as safe.
Countermeasure against chosen-insertion attacks

Use worst-case parameters for Bloom filters:

- Fix $m$, $n$ and choose $k$ that minimizes false positive probability:

$$f^{\text{adv}} = \left( \frac{nk}{m} \right)^k$$

Optimal values are:

$$k^{\text{adv}}_{\text{opt}} = \frac{m}{en} \quad \text{and} \quad f^{\text{adv}}_{\text{opt}} = e^{-m/en}$$

- Impact: On a sample Bloom filter with $m = 3200$, $n = 600$.
  - Average case: $k_{\text{opt}} = 4$, $f_{\text{opt}} = 0.077$
  - Worst case: $k^{\text{adv}}_{\text{opt}} = 2$, $f^{\text{adv}}_{\text{opt}} = 0.1$
Summary of other attacks & defenses

**Attacks:**

<table>
<thead>
<tr>
<th>Software app.</th>
<th>Attacks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scrapy: Web crawler</td>
<td>chosen-insertion, query-only</td>
</tr>
<tr>
<td>Dablooms: Spam filter</td>
<td>chosen-insertion, deletion</td>
</tr>
<tr>
<td>Squid: Web proxy</td>
<td>chosen-insertion, query-only</td>
</tr>
<tr>
<td>AIEngine: NIDS</td>
<td>query-only</td>
</tr>
<tr>
<td>sdhash: Forensic tool</td>
<td>query-only</td>
</tr>
</tbody>
</table>

**Defenses:**

- Use **HMAC**.
- Use an alternate data structure: **Bloom hash tables** [Bloom 1970]
  - Resists better to chosen-insertion attacks.
  - Is often **more memory efficient** than Bloom filters.
  - On average $O(k)$ hash computations for items not in the table.
  - On average $O(\ln k)$ for items in the table.
Related work

• **Algorithmic complexity attacks** [Crosby et al. 2003]:
  - DoS attacks against hash tables.
  - Force hash tables to operate in $O(n)$ instead of $O(1)$.
  - Similar attacks against skip-lists, regular expressions, etc.

• **Independent work on Bloom filters** [Naor et al. 2015]
  - Provide a theoretical framework.
  - Study a query-only adversary: Can only adaptively query the filter.
Privacy: Safe Browsing
Google Safe Browsing in Mozilla Firefox

Reported Attack Page!

This web page at royaldice.com has been reported as an attack page and has been blocked based on your security preferences.

Attack pages try to install programs that steal private information, use your computer to attack others, or damage your system.

Some attack pages intentionally distribute harmful software, but many are compromised without the knowledge or permission of their owners.

Get me out of here!  Why was this page blocked?

Ignore this warning
And many others
“We collect: visited web pages, clickstream data or web address accessed, browser identifier and user ID.” — WOT

“collects information including: IP address, the origin of the search ... and may share this info with a third party” — Norton

Many Safe Browsing services are privacy unfriendly by design.

“...cannot determine the real URL from the information received.” — Google

- Google seems to provide the most private service.
- Hence, focus of this work.
Google Safe Browsing: When, Why and How?

- **When**: In 2008 by Google.

- **Goals**: Protect from:
  - Phishing sites
  - Malware sites

- **How**: Easy-to-use APIs in C#, Python and PHP.

- **Methodology**: Blacklists.

- **Available in**:

- **Impact**:
  - Billions of users.
  - Detects thousands of new malicious websites per day.

- **Cloned by Yandex** as Yandex Safe Browsing.
Lookup API

- Google harvests phishing and malware URLs to feed a blacklist.
- Client checks the status using a simple **HTTP GET/POST request**: 
  sb-ssl.google.com/safebrowsing/api/lookup?example.com

Issues

- **Does not scale**: Heavy network traffic.
- **Privacy**: URLs are sent in clear.
Improving privacy using a local cache

- Communication with the server is reduced.
- Better privacy.
Google Safe Browsing API (v3): Local cache

- **Blacklists:**

<table>
<thead>
<tr>
<th>List name</th>
<th>Description</th>
<th>#Entries</th>
</tr>
</thead>
<tbody>
<tr>
<td>goog-malware-shavar</td>
<td>malware</td>
<td>317,807</td>
</tr>
<tr>
<td>googpub-phish-shavar</td>
<td>phishing</td>
<td>312,621</td>
</tr>
<tr>
<td>goog-regtest-shavar</td>
<td>test file</td>
<td>29,667</td>
</tr>
<tr>
<td>goog-unwanted-shavar</td>
<td>unwanted software</td>
<td>*</td>
</tr>
<tr>
<td>goog-whitedomain-shavar</td>
<td>unused</td>
<td>1</td>
</tr>
</tbody>
</table>

- Does not handle URLs directly, instead their SHA-256 digests.

URL: www.evil.com/

SHA-256: cc7af8a3...1918

32-bit prefix: cc7af8a3...1918

- Local cache contains prefixes.
Client’s behavior chart

User’s input

URL

Canonicalize and compute digest

Found prefix?

yes

Get full digests

Found digest?

yes

Malicious URL

no

Non-malicious URL

no
Canonicalization and decompositions

• Input URL:
  \[\text{http://usr:pwd@a.b.c:port/1/2.ext?param=1#frags}\]
• \textbf{Canonicalize(Input URL)} \rightarrow \text{http://a.b.c/1/2.ext?param=1}
• Canonicalization for privacy too: Removes username and password.

• \textbf{Multiple decompositions are checked} for a single URL.

<table>
<thead>
<tr>
<th>Decompositions of canonicalized URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. a.b.c/1/2.ext?param=1</td>
</tr>
<tr>
<td>2. a.b.c/1/2.ext</td>
</tr>
<tr>
<td>3. a.b.c/1/</td>
</tr>
<tr>
<td>4. a.b.c/</td>
</tr>
<tr>
<td>5. b.c/1/2.ext?param=1</td>
</tr>
<tr>
<td>6. b.c/1/2.ext</td>
</tr>
<tr>
<td>7. b.c/1/</td>
</tr>
<tr>
<td>8. b.c/</td>
</tr>
</tbody>
</table>

• Each matching prefix is sent to the server.
• Any matching full digest \(\Rightarrow\) Initial URL is malicious.
Purpose of computing decompositions

Memory saving:

- A domain which hosts only malicious URLs.
- **Naive blacklisting**: Include all malicious prefixes in the local cache.
- **Memory-efficient blacklisting**: Include only the domain prefix.

A more intricate example:

- **Naive blacklisting**: Include a.b.c/1, a.b.c/2 and d.b.c.
- **Memory-efficient blacklisting**: Include only a.b.c and d.b.c.
Privacy of Google Safe Browsing

“Google cannot determine the real URL from the information received.” — Google Safe Browsing v3 privacy policy

Our goal: A privacy analysis of Google and Yandex Safe Browsing

<table>
<thead>
<tr>
<th>URL</th>
<th>Prefix</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.evil.com/">www.evil.com/</a></td>
<td>cc7af8a3</td>
</tr>
<tr>
<td><a href="http://www.example-1.com/11893474">www.example-1.com/11893474</a></td>
<td>cc7af8a3</td>
</tr>
<tr>
<td><a href="http://www.example-2.com/5234456210">www.example-2.com/5234456210</a></td>
<td>cc7af8a3</td>
</tr>
<tr>
<td><a href="http://www.example-3.com/616445242">www.example-3.com/616445242</a></td>
<td>cc7af8a3</td>
</tr>
</tbody>
</table>

- Privacy due to anonymity-set.
- Estimate the anonymity-set size.
- Does it suffice to have a large anonymity-set?
Our assumptions:

- Google and Yandex have incentives to behave maliciously.
- Wish to learn whether a user visits some selected URLs.

How Google can track Safe Browsing users?

- Builds a list of prefixes to track.
- Includes these prefixes in the client’s local cache.
- Learns from the requests whether a user visited a specific URL.
- Key parameter: Anonymity-set size.
Estimating anonymity-set size

- **Anonymity-set size of a prefix**: #URLs that yield the prefix.

<table>
<thead>
<tr>
<th>Year</th>
<th>#URLs</th>
<th>#Domains</th>
</tr>
</thead>
<tbody>
<tr>
<td>2008</td>
<td>1 Billion</td>
<td>177 Million</td>
</tr>
<tr>
<td>2012</td>
<td>30 Billion</td>
<td>252 Million</td>
</tr>
<tr>
<td>2013</td>
<td>60 Billion</td>
<td>271 Million</td>
</tr>
</tbody>
</table>

- Estimate anonymity-set size: **Apply balls-into-bins model.**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>$2^{23}$</td>
<td>$2^{28}$</td>
<td>$2^{29}$</td>
<td>2700</td>
<td>3845</td>
<td>4135</td>
</tr>
<tr>
<td>32</td>
<td>232</td>
<td>6984</td>
<td>13969</td>
<td>0.04</td>
<td>0.05</td>
<td>0.06</td>
</tr>
<tr>
<td>64</td>
<td>0*</td>
<td>0*</td>
<td>0*</td>
<td>0*</td>
<td>0*</td>
<td>0*</td>
</tr>
</tbody>
</table>

0* is very close to 0.

- Domains and URLs cannot be distinguished.
- Anonymity-set size seems to be large.
### Sending multiple prefixes

**Example with two prefixes**

<table>
<thead>
<tr>
<th>Decomposition</th>
<th>Prefix</th>
</tr>
</thead>
<tbody>
<tr>
<td>petsymposium.org/2016/cfp.php</td>
<td>0xe70ee6d1</td>
</tr>
<tr>
<td>petsymposium.org/2016/</td>
<td>0x1d13ba6a</td>
</tr>
<tr>
<td>petsymposium.org/</td>
<td>0x33a02ef5</td>
</tr>
</tbody>
</table>

**Intuitively:**

- Prefix for petsymposium.org/ is not enough for re-identification.
- Sending **two 32-bit prefixes**, for petsymposium.org/ and petsymposium.org/2016/ \(\approx\) sending one 64-bit prefix.
- The maximum **anonymity-set size for 64-bit prefixes is 1.** ⇒ Should lead to re-identification.
Ambiguity on two prefixes

- More than two distinct URLs may yield the same two prefixes.
- Consider a user visiting `a.b.c` with prefixes `A` and `B` in local cache.

<table>
<thead>
<tr>
<th>URL</th>
<th>Decomposition</th>
<th>Prefix</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target URL</td>
<td>a.b.c</td>
<td>a.b.c/ A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>b.c/ B</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Ambiguity</th>
<th>URL</th>
<th>Decomposition</th>
<th>Prefix</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type I</td>
<td>g.a.b.c</td>
<td>g.a.b.c/ A</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>a.b.c/ B</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>b.c/</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>← collision on 32 bits</td>
<td></td>
</tr>
<tr>
<td>Type II</td>
<td>g.b.c</td>
<td>g.b.c/ A</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>b.c/</td>
<td>B</td>
</tr>
<tr>
<td></td>
<td></td>
<td>← collision on 32 bits</td>
<td></td>
</tr>
<tr>
<td>Type III</td>
<td>d.e.f</td>
<td>d.e.f/ A</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>e.f/</td>
<td>B</td>
</tr>
<tr>
<td></td>
<td></td>
<td>← collision on 32 bits</td>
<td></td>
</tr>
</tbody>
</table>

- $\mathbb{P}[$Type III$] = 1/2^{64}$.
- Type II URLs exist only when \#decomp. on the domain $> 2^{32}$.
- $\mathbb{P}[$Type I$] > \mathbb{P}[$Type II$] > \mathbb{P}[$Type III$]$.
- Mainly, only Type I URLs create ambiguity in re-identification.
Goal: Identify users interested in PETs.

- Target URL is petsymposium.org/2016.
How to track a given URL: A real-world example (II)

- Target URL has Type I ambiguity with: cfp.php, link.php, faqs.php

<table>
<thead>
<tr>
<th>Decomposition</th>
<th>Prefix</th>
</tr>
</thead>
<tbody>
<tr>
<td>petsymposium.org/2016/faqs.php</td>
<td>0xaec10b3a</td>
</tr>
<tr>
<td>petsymposium.org/2016/</td>
<td>0x1d13ba6a</td>
</tr>
<tr>
<td>petsymposium.org/</td>
<td>0x33a02ef5</td>
</tr>
</tbody>
</table>

- Including 2 prefixes in the local cache ⇒ Anonymity set size of 4.
- To remove any ambiguity:
  - Need to include 3 additional prefixes for cfp.php, link.php, faqs.php.
  - A total of 5 prefixes.
- Server receives 2 prefixes ⇒ visited page is the target URL.
- Server receives 3 prefixes ⇒ visited page is either of the leaf URLs.
- The third prefix decides which leaf URL was visited.
- Generalizable to any number of prefixes.
Examples of URLs creating multiple hits

- Over 1300 such URLs distributed over 30 domains.
- More frequent in Yandex than in Google Safe Browsing.

<table>
<thead>
<tr>
<th>URL</th>
<th>matching decomposition</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://wps3b.17buddies.net/wp/cs_sub_7-2.pwf">http://wps3b.17buddies.net/wp/cs_sub_7-2.pwf</a></td>
<td>17buddies.net/wp/cs_sub_7-2.pwf</td>
</tr>
<tr>
<td></td>
<td>17buddies.net/wp/</td>
</tr>
<tr>
<td><a href="http://www.1001cartes.org/tag/emergency-issues">http://www.1001cartes.org/tag/emergency-issues</a></td>
<td>1001cartes.org/tag/emergency-issues</td>
</tr>
<tr>
<td></td>
<td>1001cartes.org/tag/</td>
</tr>
<tr>
<td><a href="http://fr.xhamster.com/user/video">http://fr.xhamster.com/user/video</a></td>
<td>fr.xhamster.com/</td>
</tr>
<tr>
<td><a href="http://nl.xhamster.com/user/video">http://nl.xhamster.com/user/video</a></td>
<td>nl.xhamster.com/</td>
</tr>
<tr>
<td><a href="http://m.wickedpictures.com/user/login">http://m.wickedpictures.com/user/login</a></td>
<td>m.wickedpictures.com/</td>
</tr>
<tr>
<td><a href="http://m.mofos.com/user/login">http://m.mofos.com/user/login</a></td>
<td>m.mofos.com/</td>
</tr>
<tr>
<td><a href="http://mobile.teenslovehugecocks.com/user/join">http://mobile.teenslovehugecocks.com/user/join</a></td>
<td>mobile.teenslovehugecocks.com/</td>
</tr>
<tr>
<td></td>
<td>teenslovehugecocks.com/</td>
</tr>
</tbody>
</table>

- Including a single prefix for xhamster.com/ blacklists both fr.xhamster.com/ and nl.xhamster.com/.
- No need to add additional prefix for French or Dutch version.
Responsible disclosure and impact

- Disclosure to Mozilla Firefox:
  
  "We have long assumed (without the math to back it up) that if Google were evil it could seed the list with prefixes that allowed it to detect whether a few users visited a few select targets." — Mozilla Firefox

- Disclosure to Yandex:
  
  "We can’t promise but we plan to study them and provide you with our feedback." — Yandex Safe Browsing team

- Non-disclosure agreement with Google.

  "Google does learn the hash prefixes of URLs, but the hash prefixes don’t provide much information about the actual URLs." — Google Safe Browsing v4 privacy policy
Conclusions & Future Work
Lesson learnt: Collisions are hard to tame in security and privacy.

Bloom filters:

- Developers tend to ignore the worst-case of algorithms.
- Data structures with ad-hoc crypto primitives are at the best risky.
- Need of secure instantiations, e.g., as in Count-Min sketches.

Safe Browsing:

- Re-establish the weakness of anonymity-set privacy model.
- Google and Yandex both employ the same privacy model:
  - Google is privacy aware.
  - Yandex less so.
Future work

Bloom filters:

- **On Bloom filters**: Bloom paradox [Rottenstreich 2015].
- **Beyond Bloom filters**: Security of Bloom filter variants.

Safe Browsing:

- **Accountability**: Need of a decentralized blacklist management system [Freudiger et al. 2015].
- **Privacy**: Can local cache improve Private Information Retrieval?
Thank you!
Other works not covered today

- Performance of cryptographic accumulators.
- Private password auditing.
- (In)Security of Google and Yandex Safe Browsing.
- Alerting websites: Risks and solutions.
- Decompression quines and anti-viruses.
- Pitfalls of hashing for privacy.
- Linkable (zero-knowledge) proofs for private and accountable gossip.