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

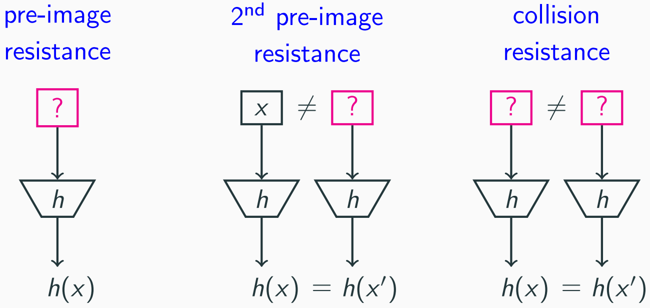
January 6, 2017

Privatics team, Inria
Université Grenoble Alpes

Hashing

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

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



	pre-image resistance	2nd pre-image resistance	collision resistance
Best generic attack	2^ℓ	2^ℓ	$2^{\ell/2}$

Are collisions always bad? (I)

A simple use case:

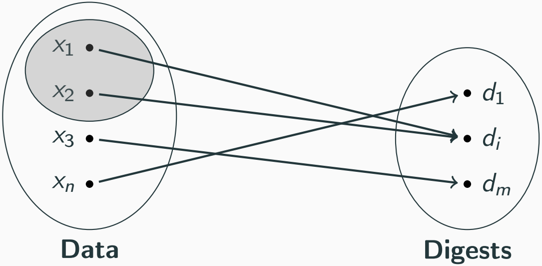
- Instead of storing n (large) data items, store their digests.
- If ℓ is large, collisions are hard to find $\Rightarrow$ space required $= n \times \ell$ bits.

Are collisions always bad? (I)

A simple use case:

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

Collisions for further space savings:



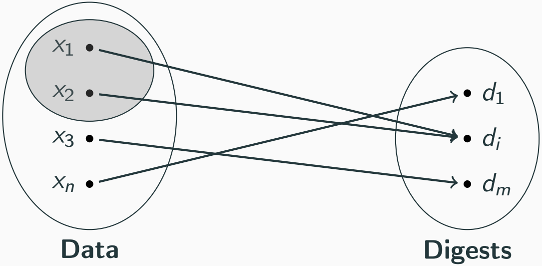
- d_i now substitutes both x_1 and x_2 $\Rightarrow$ space required $< n \times \ell$ bits.
- Caveat: May introduce some unexpected behavior.

Are collisions always bad? (I)

A simple use case:

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

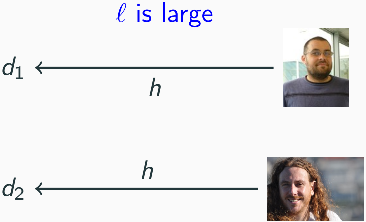
Collisions for further space savings:



- 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

Use case in privacy:

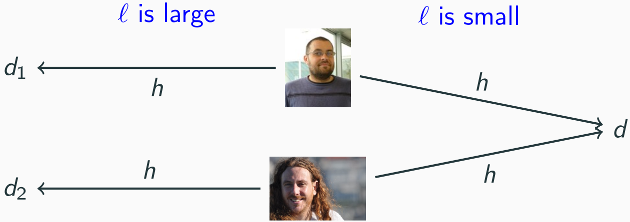
- Hashing as a pseudonymization technique.
- If ℓ is large, but #identifiers n is enumerable (in reasonable time)
    - Exhaustive search breaks pseudonymization.



Are collisions always bad? (II)

Use case in privacy:

- Hashing as a pseudonymization technique.
- If ℓ is large, but #identifiers n is enumerable (in reasonable time)
  - Exhaustive search breaks pseudonymization.



- If ℓ 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 and outline

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.

Contrasting perspectives and outline

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 $\vec{z}$ of size m compressing a set of n items.
- k uniform and independent hash functions: $h_i : \{0,1\}^* \to [0, m-1]$
- $\vec{z}$ initialized to $\vec{0}$.

Operations:

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

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

0	0	0	0	0	0	0	0	0	0

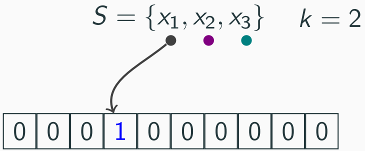
Bloom filters [Bloom 1970]

Setup(m, n, k):

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

Operations:

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



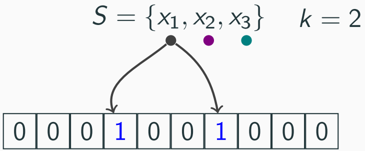
Bloom filters [Bloom 1970]

Setup(m, n, k):

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

Operations:

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



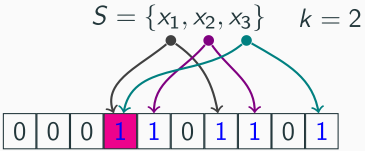
Bloom filters [Bloom 1970]

Setup(m, n, k):

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

Operations:

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



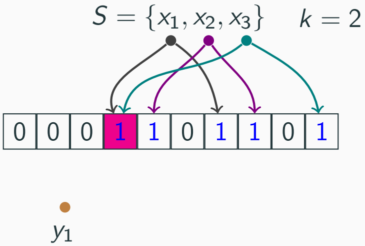
Bloom filters [Bloom 1970]

Setup(m, n, k):

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

Operations:

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



- **Query(y)**: Return `True` if bits of $\vec{z}$ at $h_1(y), \ldots, h_k(y)$ are all 1.

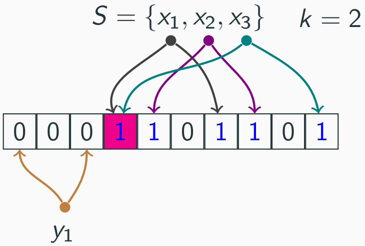
Bloom filters [Bloom 1970]

Setup(m, n, k):

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

Operations:

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



- **Query(y)**: Return `True` if bits of $\vec{z}$ at $h_1(y), \ldots, h_k(y)$ are all 1.

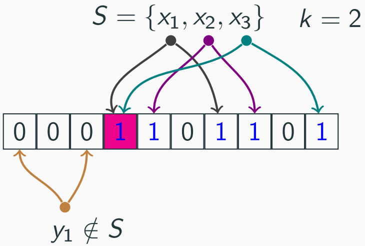
Bloom filters [Bloom 1970]

Setup(m, n, k):

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

Operations:

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



- **Query(y)**: Return `True` if bits of $\vec{z}$ at $h_1(y), \ldots, h_k(y)$ are all 1.

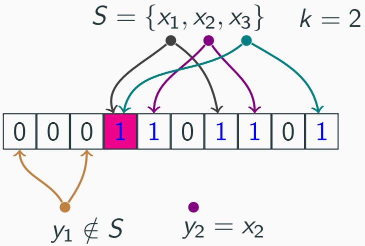
Bloom filters [Bloom 1970]

Setup(m, n, k):

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

Operations:

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



- **Query(y)**: Return `True` if bits of $\vec{z}$ at $h_1(y), \ldots, h_k(y)$ are all 1.

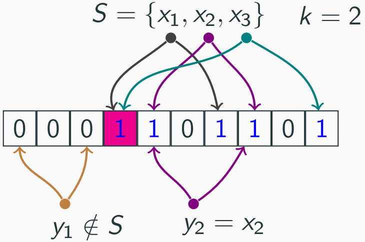
Bloom filters [Bloom 1970]

Setup(m, n, k):

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

Operations:

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



- **Query(y)**: Return `True` if bits of $\vec{z}$ at $h_1(y), \ldots, h_k(y)$ are all 1.

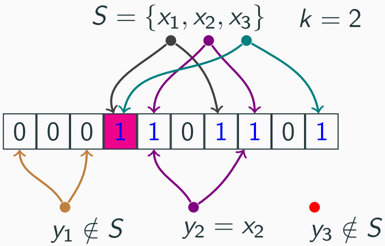
Bloom filters [Bloom 1970]

Setup(m, n, k):

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

Operations:

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



- **Query(y)**: Return `True` if bits of $\vec{z}$ at $h_1(y), \ldots, h_k(y)$ are all 1.

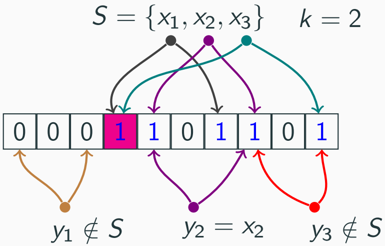
Bloom filters [Bloom 1970]

Setup(m, n, k):

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

Operations:

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



- **Query(y)**: Return `True` if bits of $\vec{z}$ at $h_1(y), \ldots, h_k(y)$ are all 1.

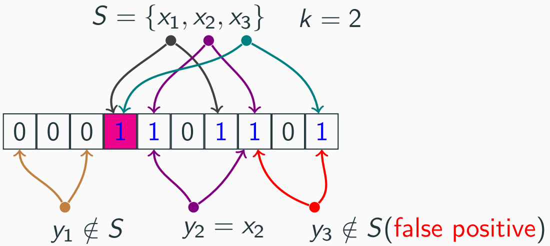
Bloom filters [Bloom 1970]

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

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

Operations:

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



- **Query(**y**)**: Return `True` if bits of $\vec{z}$ at $h_1(y), \ldots, h_k(y)$ are all 1.

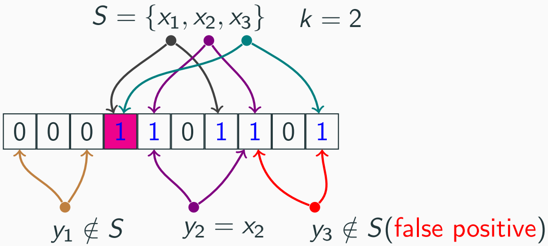
Bloom filters [Bloom 1970]

Setup(m, n, k):

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

Operations:

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



- **Query(y)**: Return `True` if bits of $\vec{z}$ 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.

Query-only adversary

Capabilities: Only queries to the filter.
Assumption: State of the filter is known.

Goals:

- Craft items that generate false positives

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.

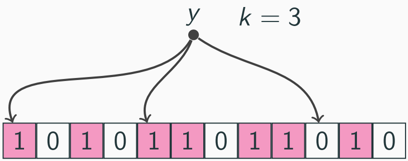
Query-only adversary

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.

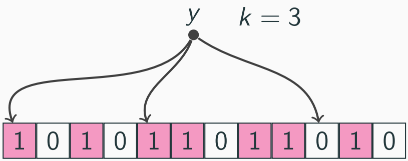


Query-only adversary

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.



  - 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.

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.

Chosen-insertion adversary

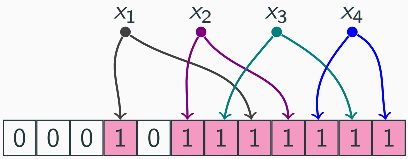
Capabilities: Can choose items to insert in the filter.

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.



Chosen-insertion adversary

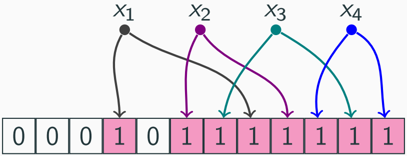
Capabilities: Can choose items to insert in the filter.

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.

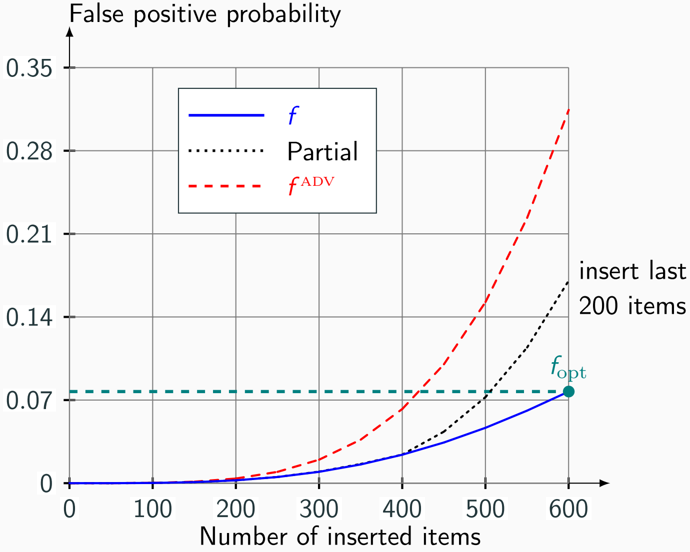


Impact:

	No attack	**Under attack**
#bits set to 1	$0.72nk_{opt}$	nk_{opt}
false positive rate (f)	$\frac{1}{2}^{k_{opt}}$	$\left(\frac{nk_{\mathbf{opt}}}{m}\right)^{k_{\mathbf{opt}}}$

Impact on a sample filter

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



Applying adversary models

Factors enabling our attacks:

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

Vulnerable software applications:

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

Bypassing a forensic tool

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:

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

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:

$$k_{\text{opt}}^{\text{adv}} = \frac{m}{en} \quad \text{and} \quad f_{\text{opt}}^{\text{adv}} = e^{-m/en}$$

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:

$$k_{\text{opt}}^{\text{adv}} = \frac{m}{en} \quad \text{and} \quad f_{\text{opt}}^{\text{adv}} = e^{-m/en}$$

- Impact: On a sample Bloom filter with $m = 3200$, $n = 600$.
  - Average case: $k_{\text{opt}} = 4$, $f_{\text{opt}} = 0.077$
  - Worst case: $k_{\text{opt}}^{\text{adv}} = 2$, $f_{\text{opt}}^{\text{adv}} = 0.1$

Summary of other attacks & defenses

Attacks:

Software app.	**Attacks**
Scrapy: Web crawler	chosen-insertion, query-only
Dablooms: Spam filter	chosen-insertion, deletion
Squid: Web proxy	chosen-insertion, query-only
AIEngine: NIDS	query-only
`sdhash`: Forensic tool	query-only

Summary of other attacks & defenses

Attacks:

Software app.	**Attacks**
Scrapy: Web crawler	chosen-insertion, query-only
Dablooms: Spam filter	chosen-insertion, deletion
Squid: Web proxy	chosen-insertion, query-only
AIEngine: NIDS	query-only
`sdhash`: Forensic tool	query-only

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 $\mathcal{O}(k)$ hash computations for items not in the table.
  - On average $\mathcal{O}(\ln k)$ for items in the table.

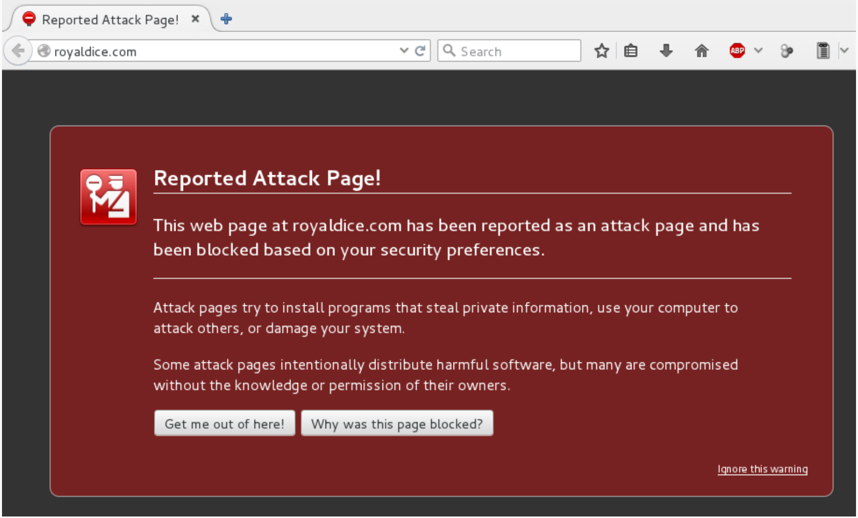
Related work

- Algorithmic complexity attacks [Crosby et al. 2003]:
  - DoS attacks against hash tables.
  - Force hash tables to operate in $\mathcal{O}(n)$ instead of $\mathcal{O}(1)$.
  - 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



And many others

Adverted privacy policy

"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.

Adverted privacy policy

"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*

Adverted privacy policy

"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`

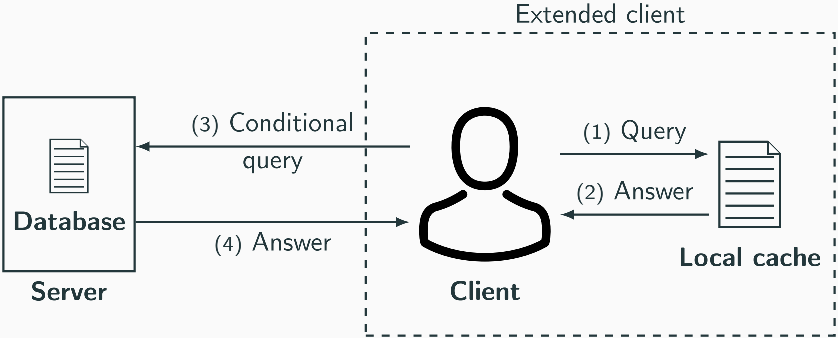
Lookup API

- 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:

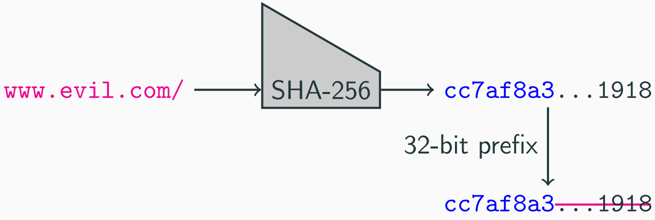
List name	Description	#Entries
goog-malware-shavar	malware	317,807
googpub-phish-shavar	phishing	312,621
goog-regtest-shavar	test file	29,667
goog-unwanted-shavar	unwanted software	*
goog-whitedomain-shavar	unused	1

Google Safe Browsing API (v3): Local cache

- Blacklists:

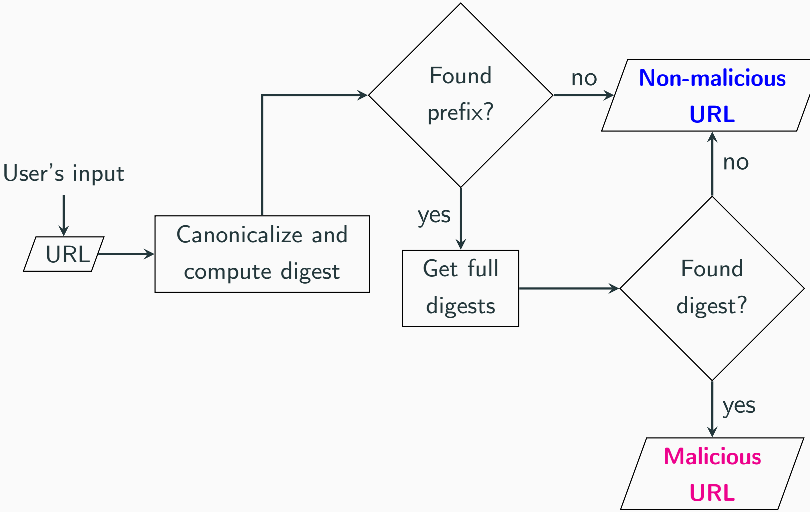
List name	Description	#Entries
goog-malware-shavar	malware	317,807
googpub-phish-shavar	phishing	312,621
goog-regtest-shavar	test file	29,667
goog-unwanted-shavar	unwanted software	*
goog-whitedomain-shavar	unused	1

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



- Local cache contains prefixes.

Client's behavior chart



Canonicalization and decompositions

- Input URL:
  `http://usr:pwd@a.b.c:port/1/2.ext?param=1#frags`
- Canonicalize(Input URL) $\rightarrow$ `http://a.b.c/1/2.ext?param=1`
- Canonicalization for privacy too: Removes username and password.

Canonicalization and decompositions

- Input URL:
  `http://usr:pwd@a.b.c:port/1/2.ext?param=1#frags`
- Canonicalize(Input URL) → `http://a.b.c/1/2.ext?param=1`
- Canonicalization for privacy too: Removes username and password.

- Multiple decompositions are checked for a single URL.

Decompositions of canonicalized URL

1. `a.b.c/1/2.ext?param=1`
2. `a.b.c/1/2.ext`
3. `a.b.c/1/`
4. `a.b.c/`
5. `b.c/1/2.ext?param=1`
6. `b.c/1/2.ext`
7. `b.c/1/`
8. `b.c/`

- Each matching prefix is sent to the server.
- Any matching full digest ⇒ Initial URL is malicious.

Purpose of computing decompositions

Memory saving:

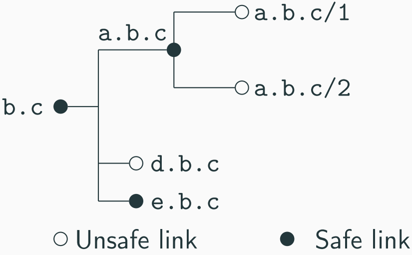
- A domain which hosts only malicious URLs.
- Naive blacklisting: Include all malicious prefixes in the local cache.
- Memory-efficient blacklisting: Include only the domain prefix.

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:

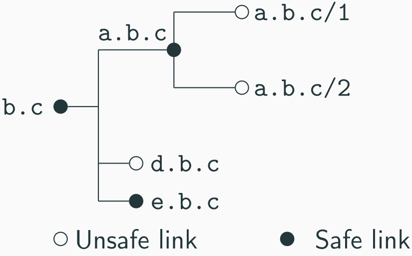


Purpose of computing decompositions

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

URL	Prefix
`www.evil.com/`	`cc7af8a3`
`www.example-1.com/11893474`	`cc7af8a3`
`www.example-2.com/5234456210`	`cc7af8a3`
`www.example-3.com/616445242`	`cc7af8a3`

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

Tracking Safe Browsing users

Our assumptions:

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

Tracking Safe Browsing users

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.

Year	#URLs	#Domains
2008	1 Billion	177 Million
2012	30 Billion	252 Million
2013	60 Billion	271 Million

Estimating anonymity-set size

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

Year	#URLs	#Domains
2008	1 Billion	177 Million
2012	30 Billion	252 Million
2013	60 Billion	271 Million

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

	Avg. for **URLs**			Avg. for **Domains**		
Prefix length (bits)	2008	2012	2013	2008	2012	2013
16	2^{23}	2^{28}	2^{29}	2700	3845	4135
32	232	6984	13969	0.04	0.05	0.06
64	0^*	0^*	0^*	0^*	0^*	0^*

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

Decomposition	Prefix
petsymposium.org/2016/cfp.php	0xe70ee6d1
petsymposium.org/2016/	0x1d13ba6a
petsymposium.org/	0x33a02ef5

Intuitively:

- Prefix for `petsymposium.org/` is not enough for re-identification.

Sending multiple prefixes

Example with two prefixes

Decomposition	Prefix
`petsymposium.org/2016/cfp.php`	`0xe70ee6d1`
`petsymposium.org/2016/`	`0x1d13ba6a`
`petsymposium.org/`	`0x33a02ef5`

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.
  $\Rightarrow$ 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.

		URL	Decomposition	Prefix	
	Target URL	`a.b.c`	`a.b.c/`	A	
			`b.c/`	B	
Ambiguity	Type I	`g.a.b.c`	`g.a.b.c/`	C	
			`a.b.c/`	A	
			`b.c/`	B	
	Type II	`g.b.c`	`g.b.c/`	A	← collision on 32 bits
			`b.c/`	B	
	Type III	`d.e.f`	`d.e.f/`	A	← collision on 32 bits
			`e.f/`	B	← collision on 32 bits

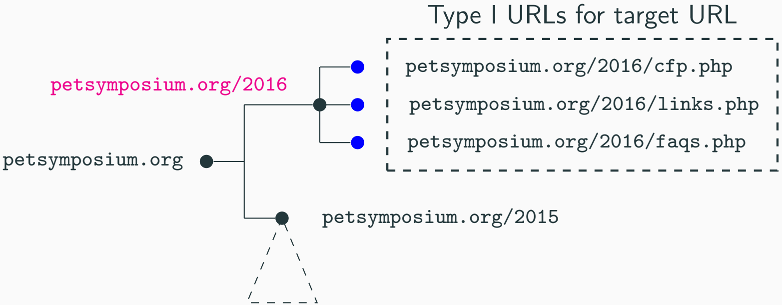
Ambiguity on two prefixes

- 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.

		URL	Decomposition	Prefix	
	Target URL	`a.b.c`	`a.b.c/`	A	
			`b.c/`	B	
Ambiguity	Type I	`g.a.b.c`	`g.a.b.c/`	C	
			`a.b.c/`	A	
			`b.c/`	B	
	Type II	`g.b.c`	`g.b.c/`	A	← collision on 32 bits
			`b.c/`	B	
	Type III	`d.e.f`	`d.e.f/`	A	← collision on 32 bits
			`e.f/`	B	← collision on 32 bits

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

How to track a given URL: A real-world example (I)



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,

Decomposition	Prefix
petsymposium.org/2016/cfp.php	0xe705b6d1
petsymposium.org/2016/	0x1d13ba6a
petsymposium.org/	0x33a02ef5

How to track a given URL: A real-world example (II)

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

Decomposition	Prefix
petsymposium.org/2016/link.php	0xdab45c01
petsymposium.org/2016/	0x1d13ba6a
petsymposium.org/	0x33a02ef5

How to track a given URL: A real-world example (II)

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

Decomposition	Prefix
`petsymposium.org/2016/faqs.php`	`0xaec10b3a`
`petsymposium.org/2016/`	`0x1d13ba6a`
`petsymposium.org/`	`0x33a02ef5`

How to track a given URL: A real-world example (II)

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

Decomposition	Prefix
`petsymposium.org/2016/faqs.php`	`0xaec10b3a`
`petsymposium.org/2016/`	`0x1d13ba6a`
`petsymposium.org/`	`0x33a02ef5`

- Including 2 prefixes in the local cache $\Rightarrow$ Anonymity set size of 4.

How to track a given URL: A real-world example (II)

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

Decomposition	Prefix
`petsymposium.org/2016/faqs.php`	`0xaec10b3a`
`petsymposium.org/2016/`	`0x1d13ba6a`
`petsymposium.org/`	`0x33a02ef5`

- Including 2 prefixes in the local cache $\Rightarrow$ 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.

How to track a given URL: A real-world example (II)

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

Decomposition	Prefix
petsymposium.org/2016/faqs.php	0xaec10b3a
petsymposium.org/2016/	0x1d13ba6a
petsymposium.org/	0x33a02ef5

- 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.

How to track a given URL: A real-world example (II)

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

Decomposition	Prefix
petsymposium.org/2016/faqs.php	0xaec10b3a
petsymposium.org/2016/	0x1d13ba6a
petsymposium.org/	0x33a02ef5

- 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.

	URL	matching decomposition
Google	http://wps3b.17buddies.net/wp/cs_sub_7-2.pwf	17buddies.net/wp/cs_sub_7-2.pwf
		17buddies.net/wp/
	http://www.1001cartes.org/tag/emergency-issues	1001cartes.org/tag/emergency-issues
		1001cartes.org/tag/
Yandex	http://fr.xhamster.com/user/video	fr.xhamster.com/
		xhamster.com/
	http://nl.xhamster.com/user/video	nl.xhamster.com/
		xhamster.com/
	http://m.wickedpictures.com/user/login	m.wickedpictures.com/
		wickedpictures.com/
	http://m.mofos.com/user/login	m.mofos.com/
		mofos.com/
	http://mobile.teenslovehugecocks.com/user/join	mobile.teenslovehugecocks.com/
		teenslovehugecocks.com/

- Including a single prefix for `xhmaster.com/` blacklists both `fr.xhmaster.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.
- Launch of Google Safe Browsing API v4 (In June 2016).

  "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

Conclusions

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.