RFC 9498: The GNU Name System

Christian Grothoff, Martin Schanzenbach and Bernd Fix

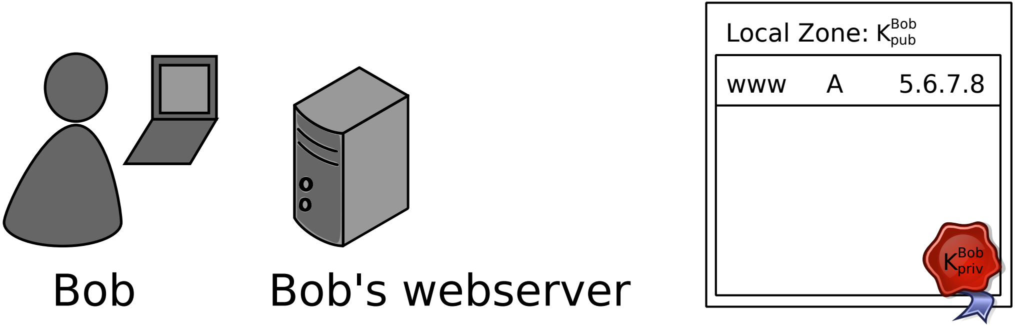
Berner Fachhochschule

22.12.2023

The GNU Name System

- Decentralized name system with secure memorable names
- Delegation used to achieve transitivity
- Also supports globally unique, secure identifiers
- Achieves query and response privacy
- Provides alternative public key infrastructure
- Interoperable with DNS

Name resolution in GNS

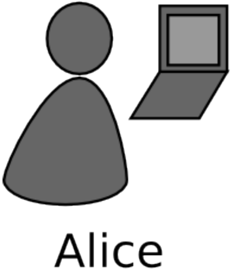


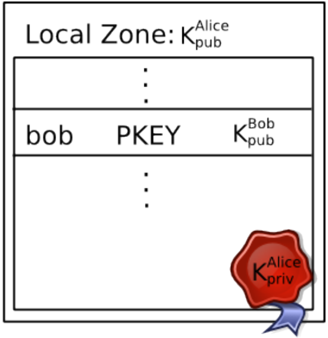
Secure introduction

Bob Builder, Ph.D.

Address: Country, Street Name 23
Phone: 555-12345
Mobile: 666-54321
Mail: bob@H2R84L4JIL3G5C

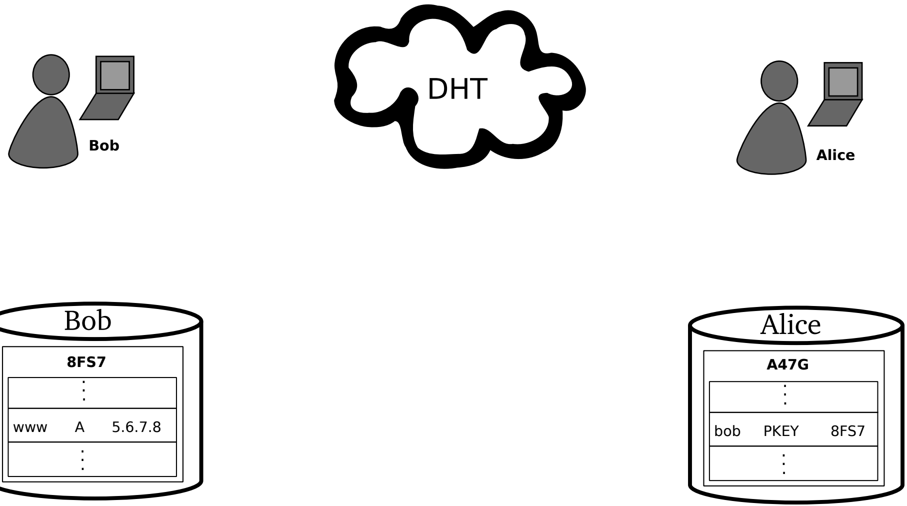
Delegation



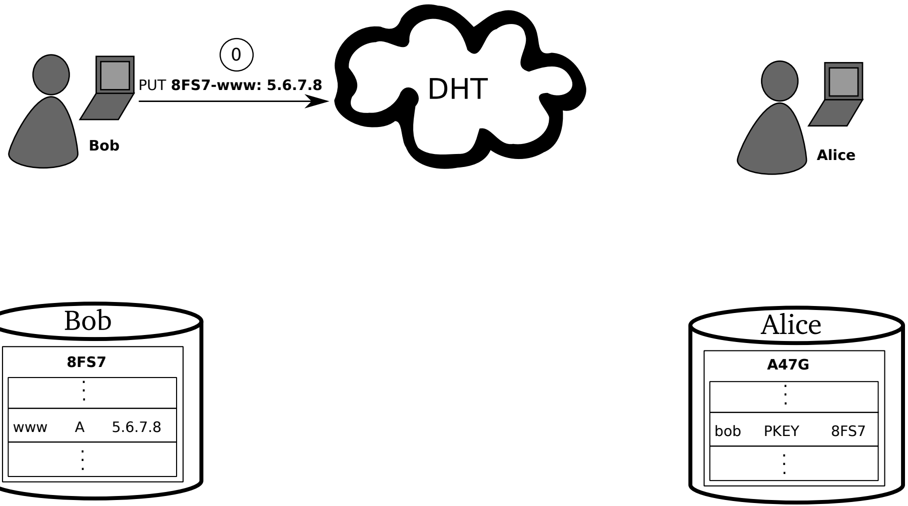


- Alice learns Bob's public key
- Alice creates delegation to zone K_{pub}^{Bob} under label **bob**
- Alice can reach Bob's webserver via **www.bob.gns.alt**

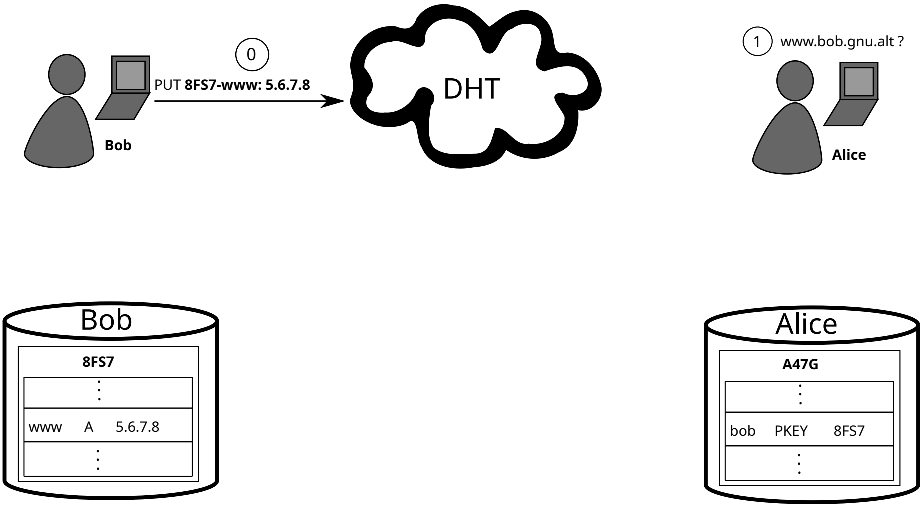
Name Resolution



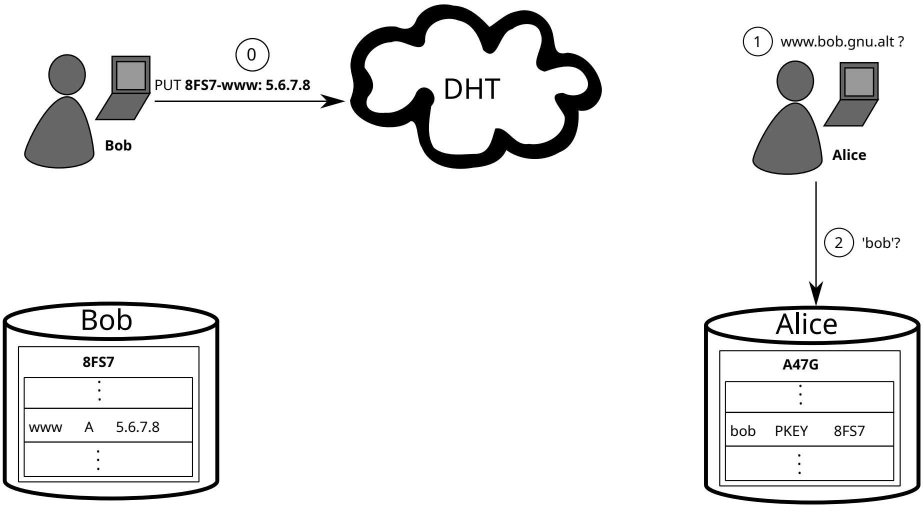
Name Resolution



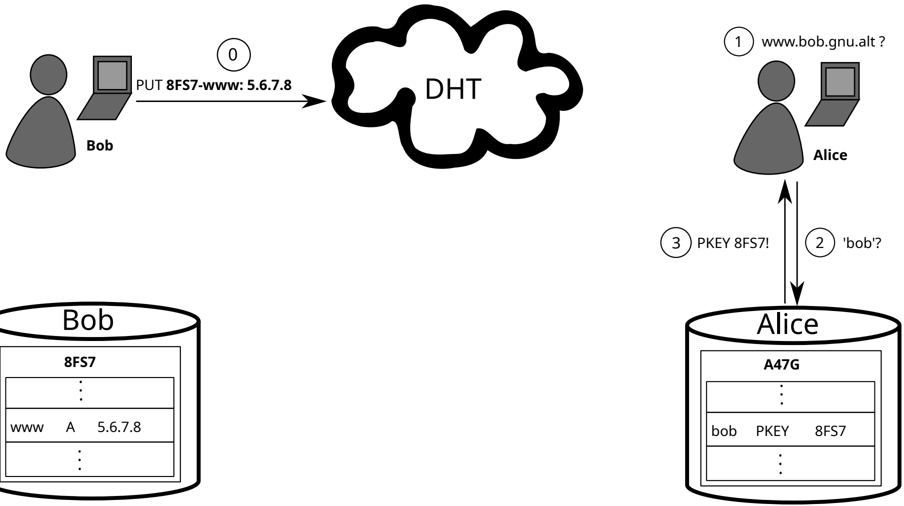
Name Resolution



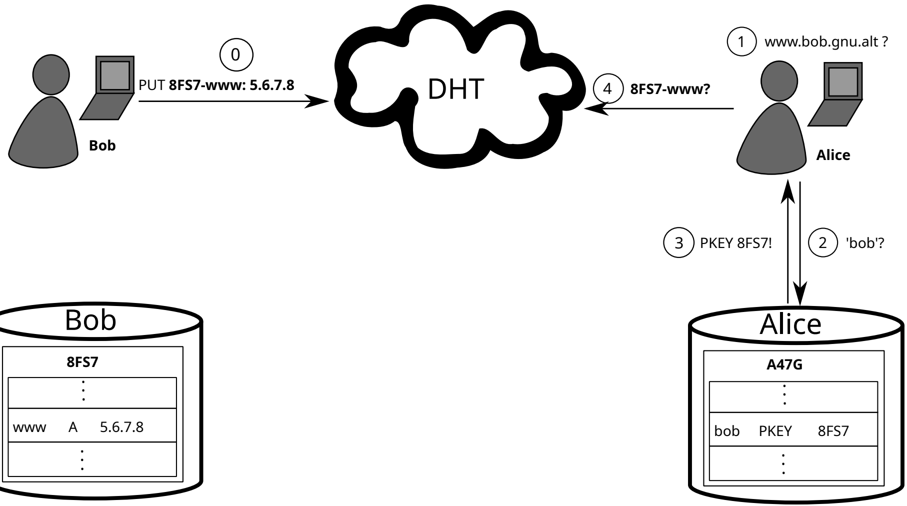
Name Resolution



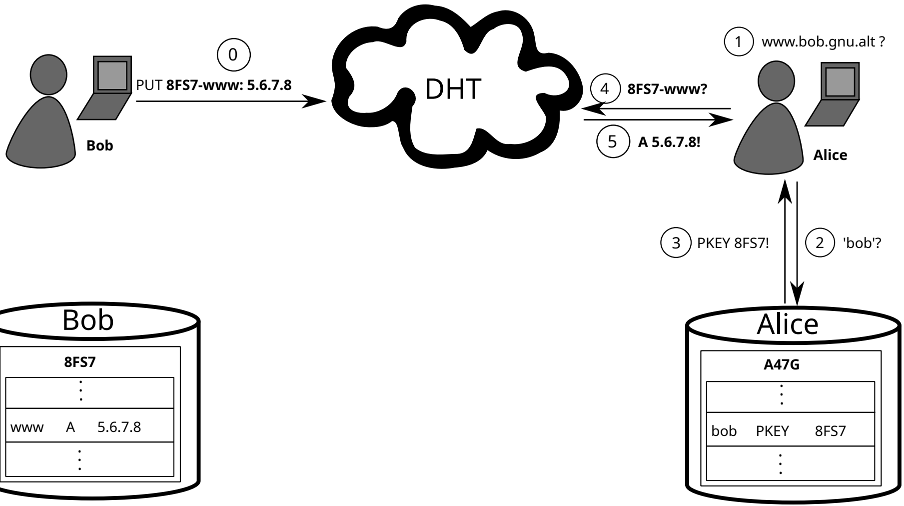
Name Resolution



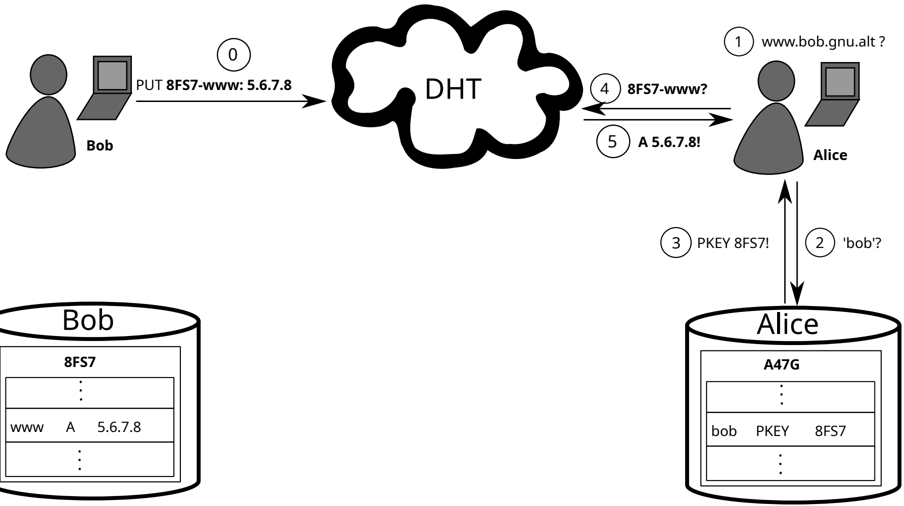
Name Resolution



Name Resolution



Privacy Issue: DHT



Query Privacy: Terminology

G	generator in ECC curve, a point
o	size of ECC group, $o := \|G\|$, o prime
x	private ECC key of zone ($x \in \mathbb{Z}_o$)
P	public key of zone, a point $P := xG$
l	label for record in a zone ($l \in \mathbb{Z}_o$)
$R_{P,l}$	set of records for label l in zone P
$q_{P,l}$	query hash (hash code for DHT lookup)
$B_{P,l}$	block with encrypted information for label l in zone P published in the DHT under $q_{P,l}$

Query Privacy: Cryptography

Publishing records $R_{P,l}$ as $B_{P,l}$ under key $q_{P,l}$

$$h := H(l, P) \quad (1)$$

$$d := h \cdot x \mod o \quad (2)$$

$$B_{P,l} := S_d(E_{HKDF(l,P)}(R_{P,l})), dG \quad (3)$$

$$q_{P,l} := H(dG) \quad (4)$$

Query Privacy: Cryptography

Publishing records $R_{P,l}$ as $B_{P,l}$ under key $q_{P,l}$

$$h := H(l, P) \tag{1}$$

$$d := h \cdot x \mod o \tag{2}$$

$$B_{P,l} := S_d(E_{HKDF(l,P)}(R_{P,l})), dG \tag{3}$$

$$q_{P,l} := H(dG) \tag{4}$$

Searching for records under label l in zone P

$$h := H(l, P) \tag{5}$$

$$q_{P,l} := H(hP) = H(hxG) = H(dG) \Rightarrow \texttt{obtain } B_{P,l} \tag{6}$$

$$R_{P,l} = D_{HKDF(l,P)}(B_{P,l}) \tag{7}$$

Key Revocation

- Certificate Revocation Lists (X.509)
- Online Certificate Status Protocol (OCSP)
- OCSP stapling (TLS)
- Publish revocations in a blockchain?

Key Revocation

- Certificate Revocation Lists (X.509)
- Online Certificate Status Protocol (OCSP)
- OCSP stapling (TLS)
- Publish revocations in a blockchain?
- **Controlled flooding**

Key Revocation via Controlled Flooding

- Revocation message signed with private key that is to be revoked
- Flooded on all links in (P2P) overlay, stored forever
- Expensive **proof-of-work** used to limit DoS-potential
- Proof-of-work can be calculated ahead of time
- Revocation messages can be computed and stored off-line if desired
- Efficient set reconciliation used when peers connect

Efficient Set Union

- Alice and Bob have sets A and B
- The sets are very large
- ... but their symmetric difference $\delta = |(A - B) \cup (B - A)|$ is small
- Now Alice wants to know $B - A$ (the elements she's missing)
- ... and Bob $A - B$ (the elements he's missing)
- How can Alice and Bob do this efficiently?
  - w.r.t. communication and computation

Simplistic Solution

- Naive approach: Alice sends A to Bob, Bob sends $B - A$ back to Alice
- ... and vice versa.

- Communication cost: $O(|A| + |B|)$ **:(**
- Ideally, we want to do it in $O(\delta)$.
- First improvement: Don't send elements of A and B, but send/request hashes.
  Still does not improve complexity **:(**

- We need some more fancy data structure!

Bloom Filters

Constant size data structure that "summarizes" a set.

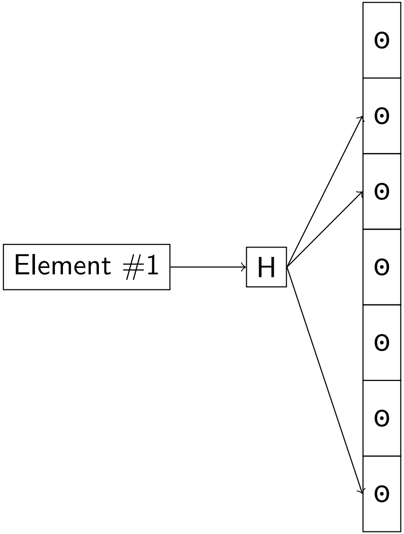
Operations:

$d = NewBF(size)$ Create a new, empty bloom filter.

$Insert(d, e)$ Insert element e into the BF d.

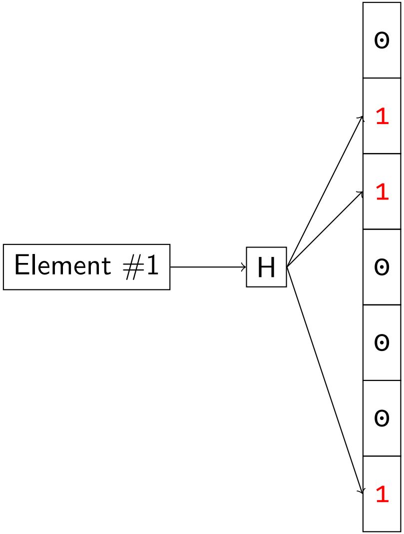
$b = Contains(d, e)$ Check if BF d contains element e.
$b \in \{$"Definitely not in set", "Probably in set"$\}$

BF: Insert



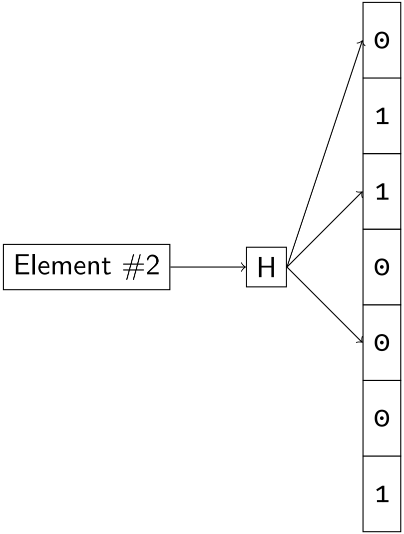
$H(\text{Element } \#1) = (2, 3, 7)$

BF: Insert



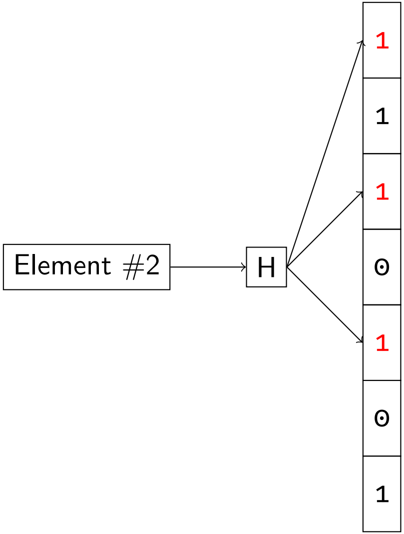
$H(\text{Element \#1}) = (2, 3, 7)$

BF: Insert



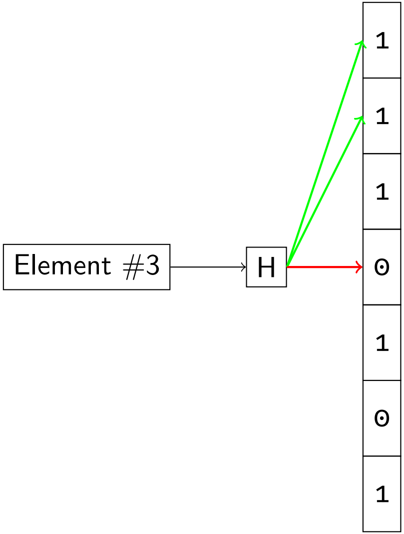
$H(\text{Element } \#1) = (2, 3, 7)$
$H(\text{Element } \#2) = (1, 3, 5)$

BF: Insert



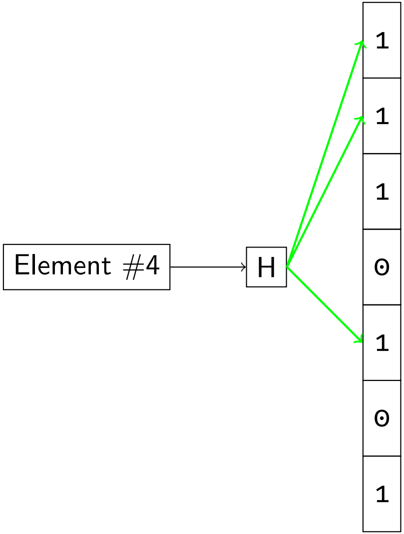
$$H(\text{Element } \#1) = (2, 3, 7)$$
$$H(\text{Element } \#2) = (1, 3, 5)$$

BF: Membership Test



$H(\text{Element } \#1) = (2, 3, 7)$
$H(\text{Element } \#2) = (1, 3, 5)$

BF: Membership Test (false positive)



$$H(\text{Element } \#1) = (2, 3, 7)$$
$$H(\text{Element } \#2) = (1, 3, 5)$$

Counting Bloom Filters

BF where buckets hold a **positive integer**.

Additional Operation:

$Remove(d, e)$ Remove element from the CBF d.

$\Rightarrow$ False negatives when removing a non-existing element.

Invertible Bloom Filters

Similar to CBF, but

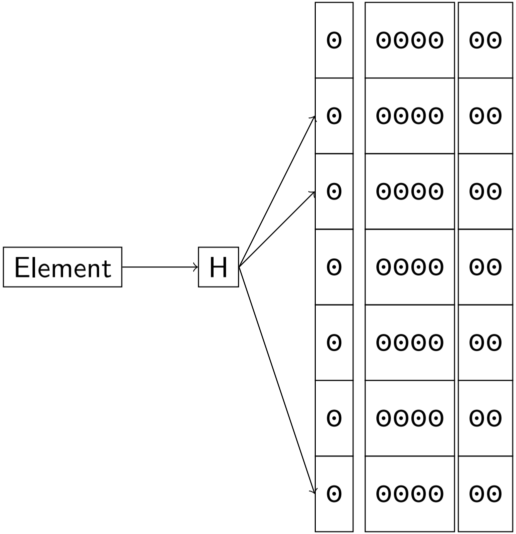
- Allow **negative counts**
- Additionaly store **(XOR-)sum of IDs (IDSUM)** in each bucket.
- Additionaly store **(XOR-)sum of hashes (XHASH)** in each bucket.

Additional Operations:

$(e, r) = Extract(d)$ Extract an element ID (e) from the IBF d, with result code $r \in \{left, right, done, fail\}$

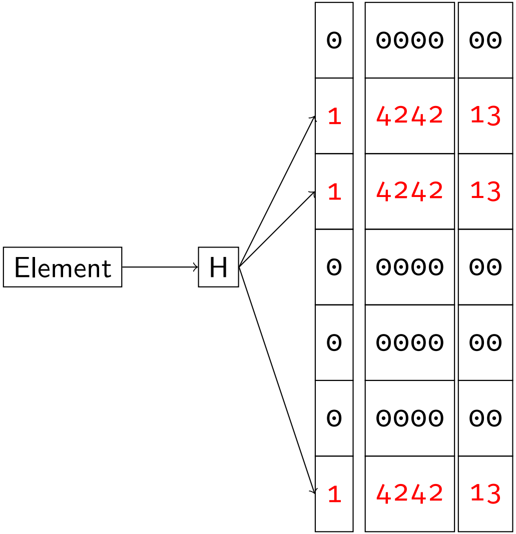
$d' = SymDiff(d_1, d_2)$ Create an IBF that represents the symmetric difference of d_1 and d_2.

IBF: Insert Element #1



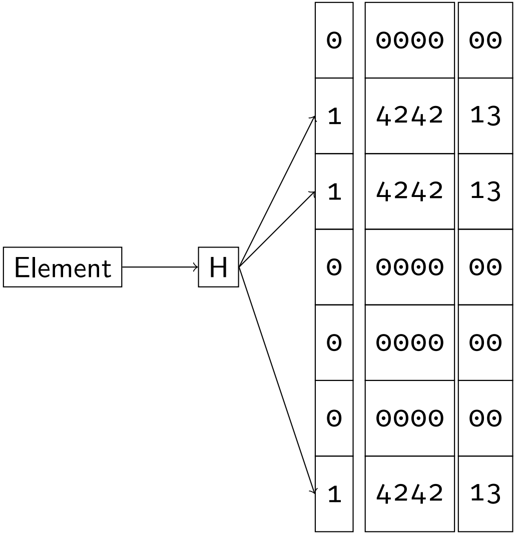
$H(\text{Element } \#1) \mapsto (2, 3, 7)$
$H'(\text{Element } \#1) \mapsto 4242 \text{ (ID)}$
$H''(4242) \mapsto 13$

IBF: Insert Element #1



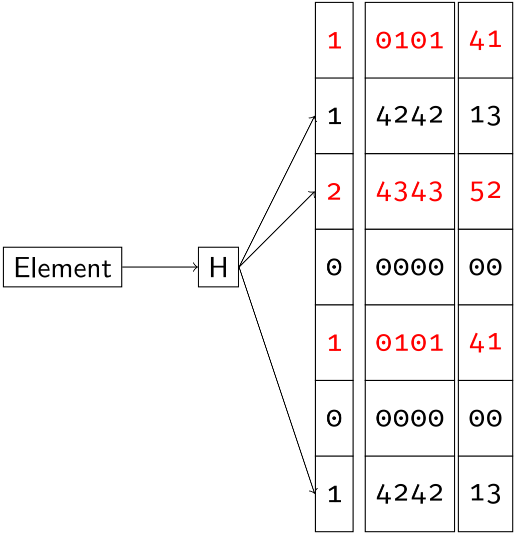
$$H(\text{Element } \#1) \mapsto (2, 3, 7)$$
$$H'(\text{Element } \#1) \mapsto 4242 \text{ (ID)}$$
$$H''(4242) \mapsto 13$$

IBF: Insert Element #2



$H(\text{Element } \#2) = (1, 3, 5)$
$H'(\text{Element } \#2) = 0101 \text{ (ID)}$
$H''(0101) \mapsto 41$

IBF: Insert Element #2



$H(\text{Element } \#2) = (1, 3, 5)$
$H'(\text{Element } \#2) = 0101 \text{ (ID)}$
$H''(0101) \mapsto 41$

Symmetric Difference on IBFs

We can directly compute the symmetric difference without extraction.

- Subtract counters
- XOR of IDSUM and XHASH values

IBF: Extract

1	0101	41	pure
1	4242	13	pure
2	4343	52	impure
0	0000	00	
1	0101	40	impure
0	0000	00	
-1	4242	13	pure

- $|counter| = 1 \land H''(IDSUM) = XHASH \Leftrightarrow$ `pure`
- Impure bucket $\Rightarrow$ potential decoding failure
- Pure bucket $\Rightarrow$ extractable element ID
- Extraction $\Rightarrow$ more pure buckets (hopefully/probably)
- Less elements $\Rightarrow$ more chance for pure buckets

The Set Union Protocol

1. Create IBFs
2. Compute SymDiff
3. Extract element IDs

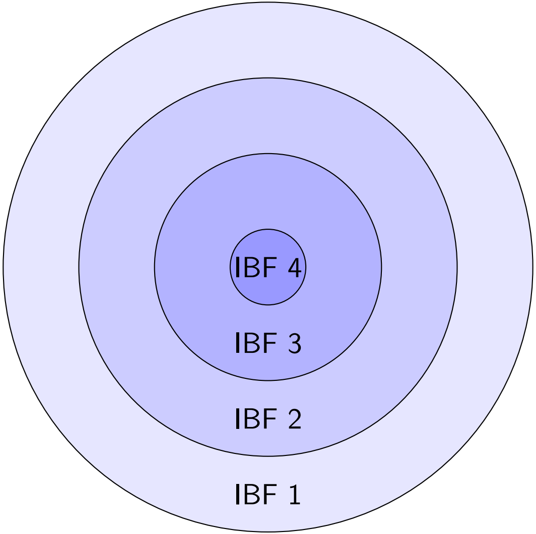
- Amount of communication and computation only depends on δ, not $|A| + |B|$:)
- How do we choose the initial size of the IBF?
- $\Rightarrow$ Do difference estimation first!

Difference Estimation

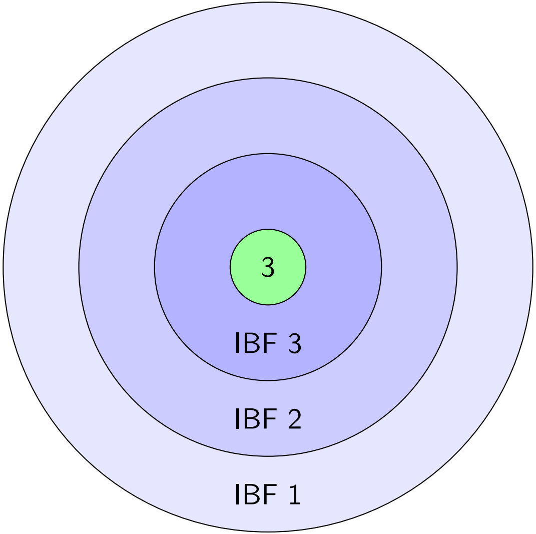
- We need an estimator that's accurate for small differences
- Turns out we can re-use IBFs for difference estimation:

1. Alice and Bob create fixed number of constant-size IBFs by sampling their set. The collection of IBFs is called a Strata Estimator (SE).
   - Stratum 1 contains $1/2$ of all elements
   - Stratum 2 contains $1/4$ of all elements
   - Stratum n contains $1/(2^n)$ all elements
2. Alice receives Bob's strata estimator
3. Alice computes $SE_{diff} = SymDiff(SE_{\mathrm{Alice}}, SE_{\mathrm{Bob}})$
   - by pair-wise *SymDiff* of all IBFs in the SE
4. Alice estimates the size of SE_{diff}.

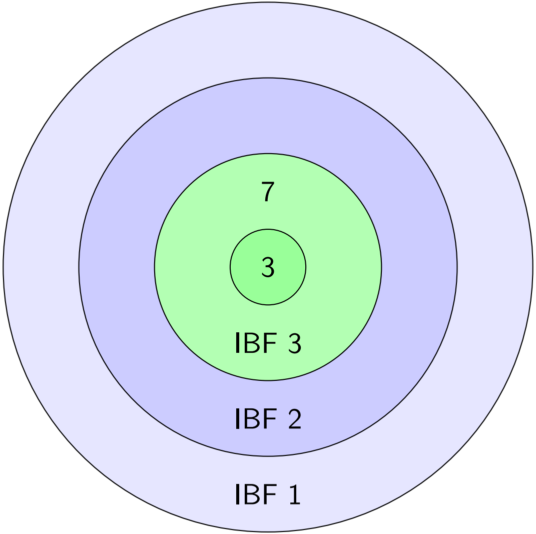
Strata Estimator



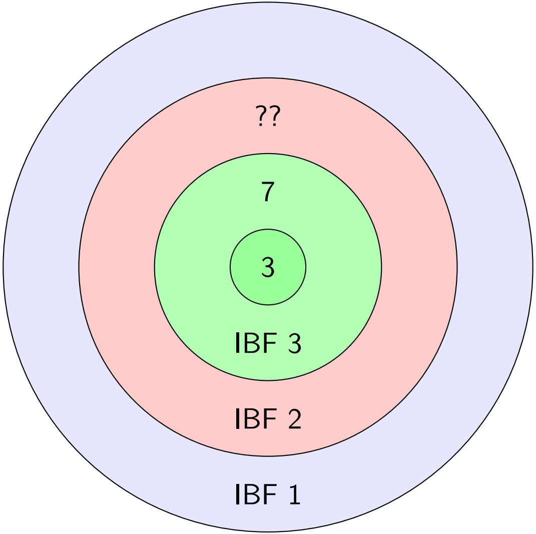
Strata Estimator



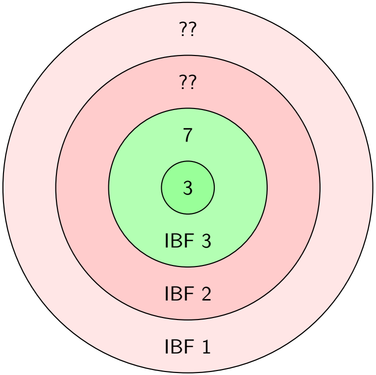
Strata Estimator



Strata Estimator



Estimation

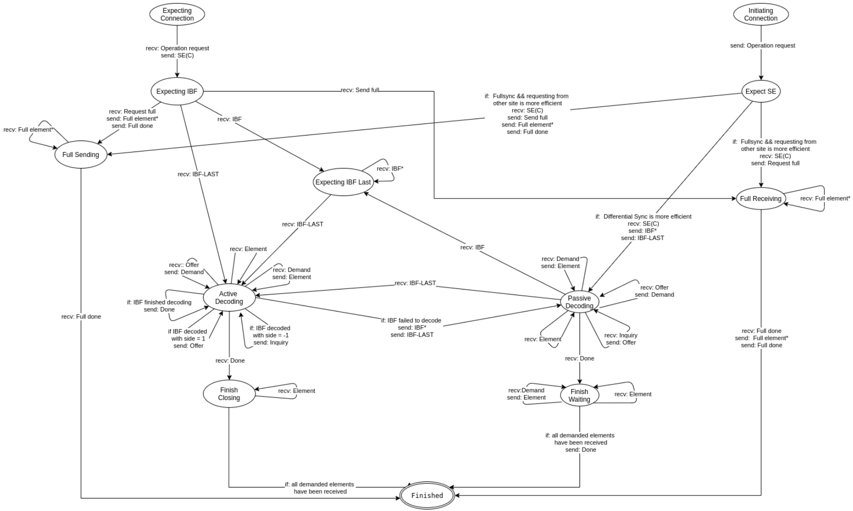


Estimate set size difference as $\frac{2^4 \cdot 3 + 2^3 \cdot 7}{2}$.

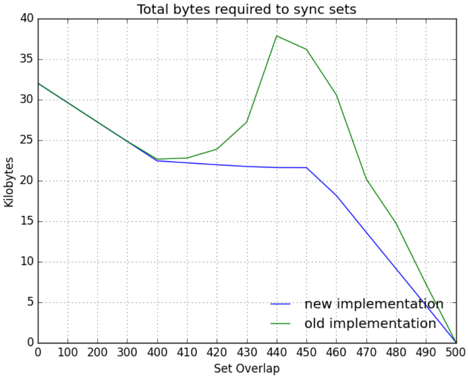
The naïve IBF Protocol

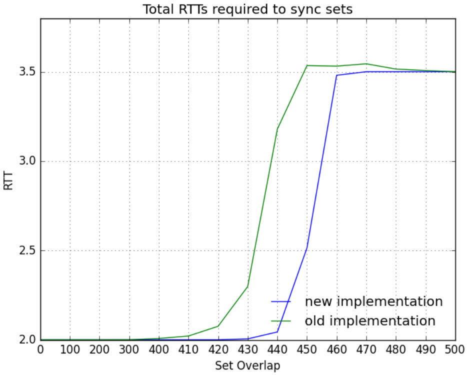
1. Alice sends SE_{Alice} to Bob
2. Bob estimates the set difference δ
3. Bob computes IBF_{Bob} for size δ and sends it to Alice
4. Alice computes IBF_{Alice}
5. Alice computes $IBF_{diff} = SymDiff(IBF_{\mathrm{Alice}}, IBF_{\mathrm{Bob}})$
6. Alice extracts element IDs from IBF_{diff}.
   - $b = left \Rightarrow$ Send element to to Bob
   - $b = right \Rightarrow$ Send element request to to Bob
   - $b = fail \Rightarrow$ Send larger IBF (double the size) to Bob, go to (3.) with switched roles
   - $b = done \Rightarrow$ We're done ...

The Complete Protocol



Implementation Performance: Tuning required!





Privacy summary

Method	Defense against MiTM	Zone privacy	Privacy vs. network	Privacy vs. operator	Traffic amplification resistance	Censorship resistance	Ease of migration
DNS	✗	✓	✗	✗	✗	✗	✓
DNSSEC	✓	✗	✗	✗	✗	✗	✗*
DNSCurve	✓	✓	✓	✗	✓	✗	✗
DNS-over-TLS	✓	n/a	✓	✗	✓	✗	✗
Namecoin	✓	✗	✓	✓	✓	✓	✗
RAINS	✓	✗	✓	✗	✓	✗	✗
GNS	✓	✓	✓	✓	✓	✓	✗

*EDNS0

Key management summary

	Suitable for personal use	Memorable	Decentralised	Modern cryptography	Understandable	Exposes metadata	Transitive
DNS	✗	✓	✗	✗	✗ (yellow)	✗	✓
DNSSEC	✗	✓	✗	✗	✗	✗	✓
DNSCurve	✗	✓	✗	✓	✗ (yellow)	✗	✓
DNS-over-TLS	✗	✓	✗	✗	✗	✗	✓
TLS-X.509	✗	✓	✗	✗	✗	✗	✓
Web of Trust	✓	✗ (yellow)	✓	✗	✗	✗	✓
TOFU	✓	✗	✓		✓	✓	✗
Namecoin	✗ (yellow)	✓	✗ (yellow)	✓	✓	✗ (yellow)	✓
RAINS	✗	✓	✗	✓	✓	✗	✓
GNS	✓	✓	✓	✓	✓	✓	✓

Conclusion

DNS	globalist
DNSSEC	authoritarian
Namecoin	libertarian
RAINS	nationalist
GNS	anarchist

In which world do you want to live?