

# Privacy-Enhanced Searches Using Encrypted Bloom Filters

Steven M. Bellovin

smb@research.att.com

AT&T Labs Research

Bill Cheswick

ches@lumeta.com

Lumeta Corp.



## Document Searches

- Organizations sometimes want to search for documents owned by another organization.
- Political or legal barriers can impede sharing (and sometimes that's good).
- Parties may be willing to share documents of demonstrable relevance — but how do you find the relevant documents?
- How do you ensure that searches are *authorized*?

## Preserving Privacy

- Even without arbitrary barriers, unrestricted information sharing can be bad.
- Potential invasion of privacy
- Again — we need a way to share selected documents, without exposing other data

## Requirements

- Querier gains no knowledge of provider's database, except for documents from valid queries. Unsuccessful queries leak no information, in either direction.
- Provider gains no knowledge of the queries
- Independent party can restrict queries
- No third party sees either queries or results

## Goal

- Solve secure indexing problem
- (Mostly) not address secure retrieval

## General Solution

- Providers create Bloom filters using a special encryption algorithm and their own key instead of the hash functions
- Queriers generate Bloom filter indices using their own keys
- A third party transforms the filter indices from the querier's key to the provider's

## Notation

- The querier is *Alice* or  $A$
- The provider is *Bob* or  $B$
- The (semi-)trusted third party is *Ted* or  $T$
- $\{X\}_k$  means “the encryption of plaintext  $X$  using key  $k$ ”



## What's a Bloom Filter?

- Efficient way to search for information if occasional false positives are acceptable
- Can only search for fixed terms, though Boolean combinations are possible and often efficient.
- False positive rate can be bounded

## Bloom Filters

- Initialize an array of  $m$  bits to zero
- For each searchable “word”  $W$ , calculate  $n$  independent hash functions  $b_i = H_i(W)$  of the datum,  $0 \leq H_i(W) < m$ .
- Set array bit  $b_i$  to 1 for each  $b_i$
- To query, calculate the same hash functions; if any selected bit is 0, the word isn’t there; if all are 1s, it’s probably there.
- If the final bit array has a 1’s density of .5, the probability of a false positive is  $2^{-n}$
- For document collections, create a bit array per document; to check for membership in a collection, bitwise-OR the individual Bloom filters.



## Sample Bloom Filter

polonium	0, 1, 2, 10, 13, 47
oralloy	10, 15, 16, 26, 35, 43
beryllium	4, 6, 10, 18, 18, 20
neutron	0, 2, 11, 25, 41, 43
Goldschmidt	1, 16, 19, 28, 42, 44
Kistiakowsky	4, 4, 10, 14, 36, 44
Meitner	12, 13, 22, 25, 27, 36
Szilard	11, 16, 33, 38, 43, 43

A sample Bloom filter of 48 elements; each search word has six bits set.



## Encrypted Bloom Filters

- Simple solution: define

$$H_i(W) = \{W\}_{k_i}$$

or

$$H_i(W) = \{W \| i\}_k$$

- Hides queries and indices from outsiders, but requires shared keys, which violates our requirements
- Solution: use a *group cipher* over the encryption operation; closure is defined as follows:

$$\forall W, \forall j, k \in K, \exists h \in K \text{ such that } \{\{W\}_k\}_j = \{W\}_h$$

We use other group properties as well, such as identities and inverses



## Pohlig-Hellman Encryption

- Group ciphers are rare, and often undesirable — you can't do iterated encryption for more strength
- At least one such cipher exists: Pohlig-Hellman
- Pick a large prime  $p = 2q + 1$  where  $q$  is also prime

$$\{W\}_k = W^k \bmod p$$

- Keys must be relatively prime to  $p - 1$ , i.e., odd and not equal to  $q$
- The decryption key  $d = k^{-1}$  corresponding to  $k$  is chosen such that  $kd \equiv 1 \bmod (p - 1)$  — easily calculable using Euclid's Algorithm
- Typical ciphertext is at least 1024 bits long; take  $\lceil \log_2 m \rceil$ -bit chunks as hash values for Bloom filter



## Using Group Cipher Encryption with Bloom Filters

- Bob creates a Bloom filter for his documents using his key  $K_B$
- Alice encrypts her query using  $K_A$  and sends the query to Ted
- Ted knows the *ratio key*  $R_{A,B}$  such that

$$\{\{W\}_{K_A}\}_{R_{A,B}} = \{W\}_{K_B}$$

and uses this key to transform the query from Alice's key to Bob's

- Ted can either query Bob's filters himself, or send the transformed query back to Alice for forwarding to Bob.
- Note: the ratio key is calculable as  $K_A^{-1} \cdot K_B \bmod (p - 1)$



## Problems with the Basic Scheme

- Obvious problem: Bob knows  $K_B$  and hence knows  $K_B^{-1}$ , and can thus decrypt the query
- Solution: instead of using  $W$  for calculating filter indices, use  $G(W)$ , where  $G$  is a cryptographic hash function — such functions are not invertible
- But Bob can still do a dictionary attack, guessing at likely query words and calculating their hashes
- Solution: “salt” the query with dummies

## **A Bad Way to Salt the Query**

- Ted splits the encrypted value into an index vector, discards some of the actual entries, and adds some new random values
- Maybe Bob can't figure out which are the real ones
- But successful queries will have a much higher hit count in an inverted index to the Bloom filter

## Sample Inverted Index

0	polonium, neutron	20	beryllium
1	polonium, Goldschmidt	22	Meitner
2	polonium, neutron	25	neutron, Meitner
4	beryllium, Kistiakowsky	26	oralloy
6	beryllium	27	Meitner
10	polonium, oralloy, beryllium, Kistiakowsky	28	Goldschmidt
11	neutron, Szilard	33	Szilard
12	Meitner	35	oralloy
13	polonium, Meitner	36	Kistiakowsky, Meitner
14	Kistiakowsky	38	Szilard
15	oralloy	41	neutron
16	oralloy, Goldschmidt, Szilard	42	Goldschmidt
18	beryllium	43	oralloy, neutron, Szilard
19	Goldschmidt	44	Goldschmidt, Kistiakowsky

A subsetted query for “polonium” might yield four 1 bits. Sending random pad entries of, say, 15, 18, 27, 43, 44 doesn’t hit more than two bits for any other word.



## How Bad is It?

To achieve a 50% probability of false indices hitting even a single word, a lower bound on the number of dummy indices is

$$\frac{c}{n} \cdot m^{(1-\frac{1}{h})} \cdot h \log h$$

where  $2/3 < c < 1$  and  $h$  is the number of bits we want to hit to be convincing (result by Jeff Lagarias)

For  $h = 2$  — unrealistically low — the result is proportional to  $\sqrt{m}$ . But for any real filter, that number will be quite high.



## **A Better Form of Salt**

- Add to the real query other terms that are likely to be somewhere in Bob's document collection
- Salter have some broader knowledge of Bob's field of knowledge
- Alice has to construct these queries; Ted doesn't know Bob's key
- Note: telling a consistent set of lies is hard...

## Another Way to Hide Queries

- Bob sends his indices filters to an index server via Ted; each index set is tagged with an encrypted version of the corresponding document name.
- Ted transforms Bob's indices to the index server's key.
- Ted transforms Alice's queries to the index server's key, and sends them to the index server
- The index server returns the encrypted document names for each successful query; Alice forwards those to Bob
- Some dummy terms may still be necessary to disguise the query topic from Bob



## Warrant Servers and Censorship Sets

- A *warrant server* enforces certain restrictions on query terms.
- Instead of transforming queries to Bob's key, Ted transforms them to the warrant server's key. The warrant server deletes from the query set any unauthorized terms, and sends the result back to Ted
- The warrant server has its own Bloom filter, and does not possess a plaintext version of the legal word list. That list is used offline to construct the Bloom filter.
- Similarly, Ted can enforce a per-querier censorship filter supplied by Bob.

## Provisioning Ted with the Ratio Keys

- How does Ted get the ratio keys without seeing encryption or decryption keys?
- If the keys form an Abelian group isomorphic to the group cipher, Alice, Bob, and Ted have a three-way conversation in which  $A$  and  $B$  transmit *blinded* versions of their keys to Ted
- Ted sends Alice and Bob some random numbers; they exchange values based on these numbers and their blinding factors
- Ted can do some arithmetic to learn only the ratio
- Note: provisioning process is  $O(s^2)$  in the number of parties. Sometimes possible to use networks of third parties. Also, most parties don't act as both providers and queriers.



## Cheaters

- Can Ted, the index server, or the warrant server cheat?
- Sure — they can send back false matches, or they can delete good ones.
- But they have no knowledge of the actual data; their cheating is random.
- Sending back extra answers is detectable by Alice; deleting good answers is no worse than random interference with network traffic.
- Conclusion: *profitable* cheating appears to be infeasible.

## Multiple Keys

- Parties who provide and query can have separate keys for each function (helps with  $s^2$  problem — typically, many fewer providers)
- Queriers could have separate keys for each provider
- Censorship and warrant servers operate on key identities, not provider identities

## Full System Design

- Many possible ways to do detailed system design
- Tradeoffs will vary, depending on precise nature of trust relationships and threat models
- Example: a rogue index server could send back document names that weren't matched by any query

## Simplified Way of Protecting Document Retrieval

- Alice prepares and sends to Ted a set of triples:  
 $\langle \{G(W)\}_{K_A}, RK, F \rangle$  where  $RK$  is a randomly-generated public key and  $F$  is the real/dummy flag. This set includes salt queries; these have  $F$  set to **false**.
- Ted forwards the query to the warrant server
- For any dummy queries, and any unauthorized queries,  $RK$  is replaced by a random number. The resulting list is sent to Bob.
- For all successful queries, Bob encrypts the document in the corresponding  $RK$  and sends it back.
- Alice can only decrypt documents belonging to real, authorized, successful queries



## Performance Issues

- Pohlig-Hellman encryption is expensive, but not that much of it is happening. Besides, the querier should have lots of time.
- Transformations by Ted are just as expensive and Ted sees lots of queries, but Ted's role can be replicated.
- Generating lots of public keys might be very expensive
- Linear searches of large sets of Bloom filters is expensive, but these can be optimized by by ORing together Bloom filters for sets of documents

## Related Work

- Song, Wagner, and Perrig developed an encrypted search algorithm that is linear in the size of documents
- Boneh et al. developed a scheme for tagging messages with a few keywords
- Goh described simple encrypted Bloom filters, and many search optimizations and enhancements
- Many papers on *Private Information Retrieval* (PIR) — hiding the topic of documents retrieved.
- Our group cipher property was dubbed *universal re-encryption* by Golle et al., and *atomic proxy encryption* by Blaze et al.



## Future Directions

- Secure, anonymous peer-to-peer networks
- An index for Anderson's "Eternity Service"
- Applications to other anonymous document sharing schemes, such as Publius

## Conclusions

- We have designed a multi-provider, multi-querier encrypted search scheme, based on a novel use of cryptographic primitives
- Authorization controls are possible
- No third party sees the queries or results

## Draft Papers

http:

`//www.research.att.com/~smb/papers/bloom-encrypt.ps`

or

http:

`//www.research.att.com/~smb/papers/bloom-encrypt.pdf`

