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*?

Requirements

- Multiple queriers, multiple providers
- Querier gains no knowledge of provider's database, except for documents from valid queries
- Provider gains no knowledge of the queries
- Independent party can restrict queries
- No third party sees either queries or results

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

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.

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* such that

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

(and 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 Pohlig-Hellman Encryption for Encrypted 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

Another Way to Hide Queries

- Bob sends his Bloom filters to an index server; each filter is tagged with an encrypted version of the corresponding document name.
- 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 operates on the encrypted queries only, and does not possess a plaintext version of the legal word list. That list is constructed and encrypted offline.
- Similarly, Ted can enforce a per-querier censorship list supplied by Bob.

Provisioning Ted with the Ratio Keys

- How does Ted get the ratio keys without seeing encryption or decryption keys?
- Roughly speaking, 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.

For Further Information

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

or

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