Indexing Encrypted Data Using Bloom Filters

Claude N. Warren, Jr

January 11, 2020

Email: claude@xenei.com
Github: `https://github.com/Claudenw`
LinkedIn: `https://www.linkedin.com/in/claudewarren`
Research Gate: `https://www.researchgate.net/profile/Claude_Warren_Jr`

Overview

Goals: What are the goals

Indexing encrypted data using Bloom filters is not a new idea. There have been several published papers that explore this problem [7, 4, 2, 10]. What has changed is the introduction of multidimensional Bloom filters [5, 13] that allow fast searching of a large number of Bloom filters.

What are we attempting

- Identify encrypted documents or records that contain specific data;
- Do not require decryption of documents for searching;
- Do not "leak" data through the index;
- Do not "leak" data through queries.

What are we excluding

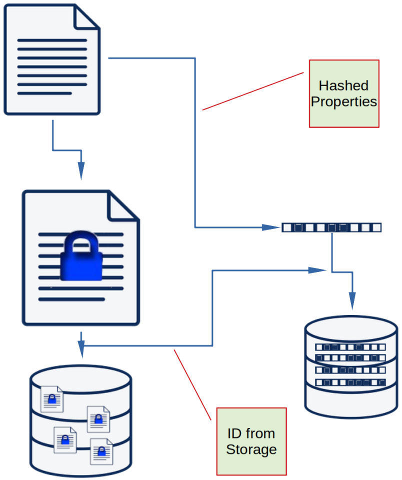
- Issues with shared keys, PKI, key distribution, etc.

The Process: Write

The write process

- Extract properties and build Bloom filter;
- Encrypt the document;
- Store encrypted document and receive storage ID;
- Store Bloom filter with encrypted storage ID in Bloom filter storage/multidimensional Bloom filter [12].

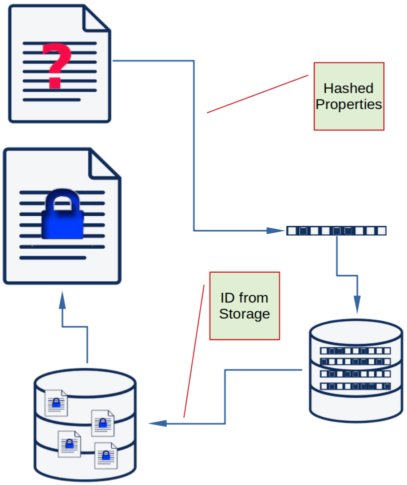
If storing 1000 filters or less a linear storage solution is more effective than a multidimensional Bloom filter [13].



Read

The read process

- Hash desired properties into a Bloom filter;
- Query the Bloom filter storage for matches and retrieve Storage ID(s);
- Read the encrypted document(s) from document storage;
- Decrypt the document(s) and filter false positives.



Introduction to the Bloom Filter: What is it

- A probabilistic data structure defined by Burton Bloom in 1970 [3];
- Constructed by hashing data multiple times;
- Can be considered as a bit vector representing a set of objects;
- Filters can be merged together: $filter2 \cup filter1$
- Membership of filter1 in the set filter2 can be checked $filter1 \cap filter2 == filter1$
- Can yield false positives but **never** false negatives.

How is it defined

Bloom filters are constrained by: the number of elements in the set, the number of hash functions, the number of bits in the vector, and the probability of false positives.

- p is the probability of false positives,
- n is the number of elements in the set represented by the Bloom filter,
- m is the number of bits, and
- k is the number of hash functions.

Mitzenmacher and Upfal [9] have shown that the relationship between these properties is:

$$p = (1 - e^{-kn/m})^k$$

Thomas Hurst provides a good calculator where the interplay between these values can be explored [8].

How is it constructed

Algorithm 1: How to construct a Bloom filter

```
Result: A populated bit vector
byte[][] buffers // the list of buffers to hash
bit[m] bitBuffer;
for buffer in buffers do
    for i=1 to k do
        long h = hash( buffer, i );
        int bitIdx = h mod m;
        bitBuffer[bitIdx] = 1;
    end
end
```

Construct a Bloom filter using Apache Commons Collections [1]

```
HashFunction hFunc = new Murmur128x86Cyclic();
Shape shape = new Shape( hFunc, 10, 1/2000000);
DynamicHasher hasher = new DynamicHasher.Builder( hFunc ).with(
    buffer ).build();  // single buffer example
BloomFilter filter = new BitSetBloomFilter( hasher, shape );
```

Data encoding issues

- Interval (decimal numeric) data does not lend itself to Bloom filter retrieval. Solution: Use ordinal values (e.g. small, medium, large) or mathematically transform the value to an integer (e.g. round decimal latitude or longitude values).
- Some properties my have similar values leading to larger number of hash conflicts. Solution: Prefix the value with the property name or abbreviation. For example if tracking automobile interior and exterior colors the values for a white care with red interior could be encoded as "exterior:white" and "interior:red".

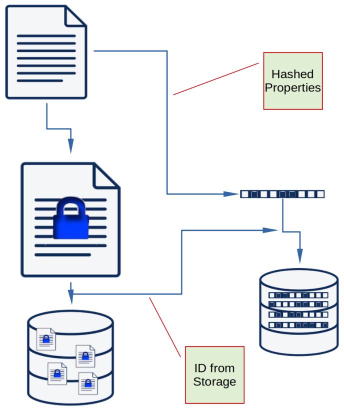
Example: GeoNames is our data

- GeoNames is a geographical database is available for download free of charge under a creative commons attribution license [6]. It contains over 25 million geographical names and consists of over 11 million unique features.
- Each object has 20 properties of which we will select: the feature code, the country code, and the first 10 names as the properties to index.
- There are only 680 unique features codes and 252 country codes defined in the GeoNames data

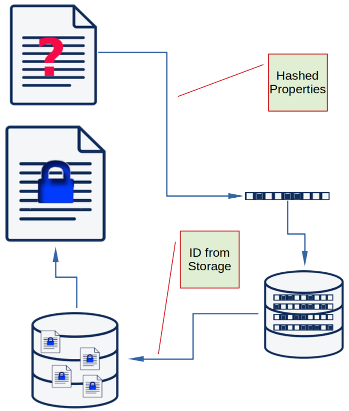
Bloom encrypted index demo

- The demo code [11] uses a multidimensional Bloom filter [5] to index 2 million (2e6) GeoName objects utilizing a 128 bit Murmur3 x86 hash implementation. The Bloom filter shape specifies: $n = 10$ and $p = 1.0/2000000$ which yields $m = 302$, $k = 21$ and $p \approx 0.0000005$ (1 in 2001957).
- The multidimensional Bloom filter library [12] uses a Hasher that does not retain the buffer bytes, just the hashed values.
- After the demo loads the data it reports that it has loaded 2 million items resulting in 704899 unique filters.
- searching for "Las Vegas" and "PPL" (GeoNames feature code for Populated Place) yields 8 results. All are named "Las Vegas" and have "PPL" designations.
- searching for "want" yields 3 results. One named "Want" and the other 2 false positives.

Demo


Hashed Properties
ID from Storage

DEMO


Hashed Properties
ID from Storage

Goals Review: Did we meet the goals

- Identify encrypted documents or records that contain specific data - Yes;
- Do not require decryption of documents for searching - Yes;
- Do not "leak" data through the index - Mostly. Brute force word encryption and analysis of Bloom filters may yield some data but some *a priori* knowledge is required;
- Do not "leak" data through queries - Mostly. Queries do leak the number of terms being searched and a brute force term encryption may be able to match them. *A priori* knowledge would help here.

Additional Info: Standard uses for Bloom filters

- Typically used where hash table solutions are too memory intensive and false positives can be addressed; for example a gating function to a longer operation. (e.g. determine if a database lookup should be made);
- In bioinformatics they are used to test the existence of a k-mer in a sequence or set of sequences. The k-mers of the sequence are indexed in a Bloom filter, and any k-mer of the same size can be queried against the Bloom filter.
- In database engines used to perform joins. A bloom filter is constructed for the one side of the join. During the join the other side values are checked against the bloom filter first.
- In the context of service discovery in a network, Bloom filters have been used to determine how many hops it is from a specific node to a node providing a desired service.
- Bloom filters are often used to search large chemical structure databases. The properties of the atom are encoded into Bloom filters that are then stored in a multidimensional Bloom filter.

What is a multidimensional Bloom filter

A multidimensional Bloom filter is a searchable collection of Bloom filters [5]. A simple list of Bloom filters is the simplest and most common form. However as the number of filters to search increases it becomes evident that a method of indexing is desired.

Indexing bloom filters is not trivial as the sorting algorithms that underlie most indexes require an ordinal comparison. However Bloom filter comparisons do not produce ordinal results due to the bit level intersection calculation. Bloom filter indexes are also hampered by the extreme speed of the bit level intersection calculation. Warren, et.al. [13] have shown that for fewer than 1000 entries there are no multidimensional Bloom filters that are faster than the linear search.

References I

[1] The Apache Software Foundation. *Apache Commons Collections*. Accessed on 11-Jan-2020. URL: https://commons.apache.org/proper/commons-collections/.

[2] Steven M Bellovin and William R Cheswick". *Privacy-Enchanced Searched Using Encrypted Bloom Filters"*. Accessed on 18-Dec-2019. 2004. URL: https://mice.cs.columbia.edu/getTechreport.php?techreportID=483.

[3] Burton H. Bloom. "Space/Time Trade-offs in Hash Coding with Allowable Errors"". In: *Communications of the ACM* 13.7 (July 1970), pp. 422–426.

[4] Yan-Cheng Chang and Michael Mitzenmacher. *Privacy Preserving Keyword Searches on Remote Encrypted Data*. Accessed on 18-Dec-2019. 2004. URL: https://eprint.iacr.org/2004/051.pdf.

References II

[5] Adina Crainiceanu and Daniel Lemire. *Bloofi: Multidimensional Bloom Filters*. Accessed on 11-Jan-2020. 2016. URL: https://arxiv.org/pdf/1501.01941.pdf.

[6] *GeoNames*. Accessed on 18-Dec-2019. URL: http://www.geonames.org/.

[7] Eu-Jin Goh. *Secure Indexes*. Accessed on 18-Dec-2019. 2004. URL: https://crypto.stanford.edu/~eujin/papers/secureindex/secureindex.pdf.

[8] Thomas Hurst. *Bloom filter calculator*. Accessed on 10-Nov-2019. URL: https://hur.st/bloomfilter/.

[9] Michael Mitzenmacher and Eli Upfal. *Probability and computing: Randomized algorithms and probabilistic analysis*. Cambridge, Cambridgeshire, UK: Cambridge University Press, 2005, pp. 109–111, 308. ISBN: 9780521835404.

References III

[10] Arisa Tajima, Hiroki Sato, and Hayato Yamana. *Privacy-Preserving Join Processing over outsourced private datasets with Fully Homomorphic Encryption and Bloom Filters*. Accessed on 18-Dec-2019. 2018. URL: https://db-event.jpn.org/deim2018/data/papers/201.pdf.

[11] Claude N. Warren Jr. *Bloom Encrypted Index*. Accessed on 11-Jan-2020. URL: https://github.com/Claudenw/BloomEncryptedIndex.

[12] Claude N. Warren Jr. *Multidimentional Bloom Filter Implementations*. Accessed on 11-Jan-2020. URL: https://github.com/Claudenw/MultidimentionalBloom.

[13] Claude N. Warren Jr. et al. “Naught For All: An Investigation of Bloom Filter Indexing Strategies”. unpublished paper. 2020.