UltraFilter: A Privacy-Preserving Bloom Filter

Kenneth Odoh
https://kenluck2001.github.io
kenneth.odoh@gmail.com

ABSTRACT

Recent work by Reviriego et al. [12] has shown susceptibility to privacy leaks for elements stored in the Bloom Filter due to the possibility of recovery attacks despite the indirection (unlinkability) afforded by the underlying hash functions. As a result, we introduced a data structure known as UltraFilter to allow for approximate set membership checks without the pre-existing privacy issues in the standard Bloom filter by employing differential privacy. We provide a threefold contribution: First, we achieve tunable privacy by modifying the noise, ε, to adjust the reconstruction error that trades off utility for privacy protection in UltraFilter. Second, we provide a set of proofs to validate our formulation and a working implementation [1] for easy replication. Finally, we demonstrate the usefulness of our formulation in an ablation study.

CCS CONCEPTS

• **Computing methodologies → Security and Privacy**.

KEYWORDS

Cryptography, Privacy, Security, Data mining, Bloom Filter , anonymity

ACM Reference Format:
Kenneth Odoh. 24. UltraFilter: A Privacy-Preserving Bloom Filter. In *Proceedings of Sydney '25: The Web Conference (TheWebConf '25).* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

The Bloom Filter is a succinct data structure well-suited for applications requiring quick lookups with compact storage. Similarly, the Bloom filter has attractive properties, including efficient data representation and versatility. Hence, this data structure has applications in caching, database management, and other areas. Our work differs from private membership testing [10] related to private information retrieval [4], leaking data in client-server settings. In our construction, the internal bit-array gets randomized to reduce the linkability of stored elements in the bit-array where privacy leaks have been demonstrated [12] despite using a one-way hashing function in the Bloom filter. UltraFilter ($\hat{BF}$) is a construction that resolves the privacy limitations of standard Bloom filters by incorporating differential privacy.

[1]**Source code**: https://github.com/kenluck2001/miscellaneous/tree/master/src/bloom

TheWebConf '25, April 28- May 2, 2025, Australia
© 2024 Association for Computing Machinery.
ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

1.1 Contributions

We have provided an implementation of UltraFilter and Standard Bloom filters as part of this manuscript, along with empirical evaluation. Our work is validated by mathematical proof showing the theoretical bounds on the pairwise distance between item pairs formulating a closed-form formulation of the reconstruction error, which allows for a tunable level of achievable privacy protection required to mitigate the privacy vulnerability discussed in Reviriego et al. [12] as shown in Definitions 6.1, 6.2. Furthermore, we have demonstrated in Claim 1 that the item insertion algorithm in UltraFilter is an ϵ-differential privacy scheme. Then, we provide theoretical upper bounds on the false positive and false negative rates for UltraFilter and Standard Bloom Filter in Subsection 6.3.

1.2 Differential Privacy

Add noise, ε, to obfuscate data, guaranteeing individual privacy even with public information release as shown in Definition 1.1.

Definition 1.1. (Differential Privacy) Following Definition 7 [7], for every pair of the datasets (D, D'), noise, ε, and a randomizer, $\mathcal{M}$ satisfies $\mathrm{P}(\mathcal{M}(D) \in \mathcal{O}) \leq e^{\varepsilon}\mathrm{P}\left(\mathcal{M}\left(D'\right) \in \mathcal{O}\right)$

1.3 Bloom Filter

The Bloom filter (BF) [2] is a probabilistic data structure that supports efficient membership testing, insertion, and lookup operations. While the standard Bloom filter lacks efficient deletion support due to the cost of rebuilding the filter, it offers compact storage and efficient query times. Standard Bloom filter (BF) has an m-sized bit-array with each bit position for storing elements, n is the maximum storage capacity, and l is the number of independent hashes. H_i sampled from a family of hash functions with a domain $[0, m-1]$ with $\forall i \in 1 \ldots l$. The supported operations: Insertion, Lookup, and Randomization are in Algorithms 1, 2, and 3.

2 RELATED WORK

The set of algorithms supported by the Standard Bloom Filter [2] lacks a mechanism for efficient deletion. Subsequent research resulted in the development of Cuckoo Filter [8] data structure that provides deletion in addition to insertion and lockup already supported in the Standard Bloom Filter. Standard Cuckoo Filter uses a fixed bit-array size for storing elements. In contrast, Dynamic Cuckoo Filter [3] adopts a variable-sized array for increased data management flexibility. Xor Filter [11] is a variant of the bloom filter that provides superior computational efficiency in terms of space and time complexity compared to the standard bloom [2] and Cuckoo filters [3]. Hybrid data structures can be constructed using a standard Bloom filter (or other variants) as an auxiliary structure for building higher-level data structures. One such example is the multidimensional Bloom filter [5], which supports the hierarchical querying of stored items.

The construction and design of the Bloom filter is an active area of research with works such as the invertible Bloom filter [9] that allows key-value construction for better flexibility in data management of stored elements. Many Bloom filters have privacy vulnerabilities that motivated Reviriego et al. [12] to show how the privacy vulnerability of the Bloom filter can influence the success of reconstruction attacks in their work. As a result, our work focuses on mitigating these privacy issues by employing differential privacy in the design of the UltraFilter. Another related development [6] has added a time-decaying constraint for storing elements in the Bloom filter. Furthermore, another work [1] on private bloom filter shares similar by utilizing flipping probabilities for randomizing bit arrays on item insertion. Our work differs by providing a unique parametrization of parameters on a sigmoid function. Furthermore, we have shown simplified mathematical proofs and robust empirical evaluations to highlight the technical characteristics of UltraFilter.

3 EXPERIMENT

We generated millions of random strings of 50 characters each. The choice of text size is arbitrarily chosen and standardized throughout our experiments. As a result, we created a balanced dataset with an equal pair of positive (items saved in bloom filter) and negative (items not stored in bloom filter) set for every experiment in this work. We displayed the technical characteristics of the bloom filter in Tables 1, 2, 3,and 4.

The primary evaluation metric for our experiments is the F1-score utilized due to its ability to measure combined precision and recall. UltraFilter (Bloom filter) exhibits high performance characteristics when F1-score $\approx$ 1. On the contrary, the data structure reflects poor performance dynamics when the F1-score $\approx$ 0. Additionally, we define the occupancy ratio, φ, as the number of elements stored divided by the total storage capacity, n, of the Bloom filter (UltraFilter).

First, we experimented to study the effect of increasing noise, ε, on UltraFilter at maximum capacity, $n = 2^{20}$, and bit-array size, $m = 2^{24}$, number of hash functions, $l = 10$, with other parameters kept constant as shown in Table 1. At a lower value of ε, we observed a reduced F1 score. Second, we analyzed the interaction effect of occupancy ratio, φ, over a range of noise, ε, on UltraFilter (Standard Bloom Filter) with performance metrics as shown in Tables 2, 3 and 4 respectively. The parameter, φ, is progressively increased with other settings and kept constant.

The mathematical notations utilized in this work are as follows: FP_{BF}, $FP_{\hat{BF}}$ for the false positive rate of standard bloom filters (BF) and UltraFilter ($\hat{BF}$) respectively. Similarly, FN_{BF} and $FN_{\hat{BF}}$ for the false negative rate of typical bloom filters (BF) and UltraFilter ($\hat{BF}$). More evaluation metrics are depicted as $F1_{BF}$, $F1_{\hat{BF}}$ for F1-score for Standard Bloom filter and UltraFilter. Furthermore, we adopt the convention for evaluation score at preset noise, ε depicted as $F1^{\varepsilon}_{\hat{BF}}$, $FP^{\varepsilon}_{\hat{BF}}$, $FN^{\varepsilon}_{\hat{BF}}$ are F1-score, false positive rate, and false negative rate at target noise, ε.

4 DISCUSSION

We have observed that when noise, $\varepsilon = 0$ (perfect secrecy), it results in worsening performance (F1-score) because sampling probability is $\frac{1}{2}$ limiting the amount of information that is storable in $\hat{BF}$.

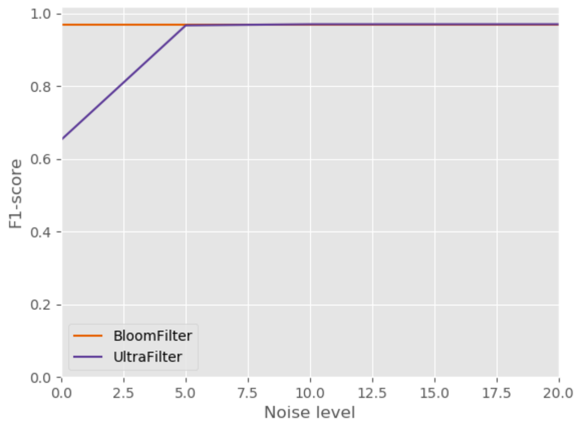

Figure 1: Noise Level, ε, versus F1-score

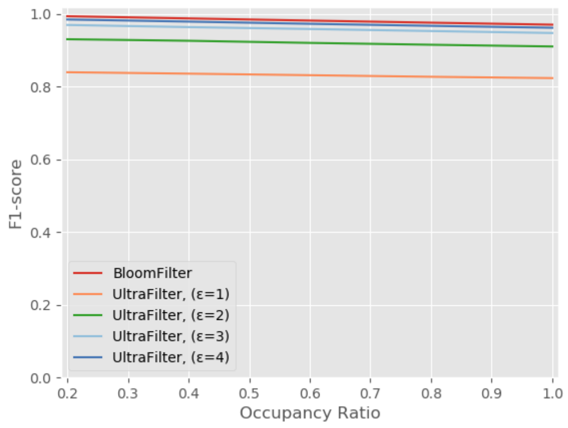

Figure 2: Occupancy ratio, φ, over noise, ε, range versus F1-score

Despite the drop in F1-score within reasonable values of ε, the performance of UltraFilter is competitive with Standard Bloom filter as it provides privacy protection as shown in Figure 1 and Table 1. We can improve the F1-score by increasing the noise, ε. However, once the threshold $\varepsilon \geq 5$ gets reached, the effect of higher noise is negligible on the F1-score due to our choice of sigmoid function for parametrizing the noise to the probabilistic bit flipping of the bit-array ($\hat{BF}$). We also noticed that the $FP_{BF} = 0.06$ and $FN_{BF} = 0$ remained constant throughout the experiment and are shown in Table 1. Based on experimental results, we suggested setting noise, $\varepsilon \in (0, 5)$ for most practical purposes.

The performance of UltraFilter degrades as the magnitude of noise, ε, becomes smaller as captured in Tables 1, 2. Similarly, the occupancy ratio, φ, is inversely proportional to the F1-score in the Standard Bloom filter and UltraFilter. This proportionality effect is observed as follows: when storing a small number of items in a large capacity Bloom filter, uncertainty is reduced, leading to a higher F1-score. However, when we are at the limits of the filter storage capacity, n, hence, the performance metric degrades as shown in Figure 2 and Table 2. False negative, $FN_{\hat{BF}}$ remain constant when the occupancy ratio, φ, is increased as other factors are kept constant in UltraFilter as seen in Table 3. False positive is directly proportional to the occupancy ratio, φ. At lower noise, ε, FP_{BF} is higher than $FP_{\hat{BF}}$ due to the squeezing effect of the sigmoid function in probabilistic flipping thereby giving rise to increased $FN_{\hat{BF}}$, whereas $FN_{BF} = 0$. Consequentially, when $\varepsilon \geq 5$, then $FP_{BF} \approx FP_{\hat{BF}}$ as described in Table 4.

Our evaluation considers the case where the bloom filter is preloaded with data and then randomized, and subsequent querying can happen. At higher noise, ε, due to smaller flipped bits, it is possible to update the bit-array ($\hat{BF}$) with new item insertions after randomization. On the contrary, at lower noise, ε, it can be challenging to update the bit-array ($\hat{BF}$) after randomization because of more flipped bits. As a result, repeated insertions can impact the reasoning between privacy losses and utility. Another option is to store the original bit-array (BF) for future copying. Subsequently, insertions are made only in the bit-array (BF) and randomized on every input to create bit-array ($\hat{BF}$). However, repeated copying of the bit-arrays (BF to $\hat{BF}$) may not be ideal for every application. Conversely, based on our design implementation, the functionality of UltraFilter is limited to ASCII [2] string, so it may not work well for languages with wide-character sets [3]. Future work would improve on this known flaw.

5 CONCLUSION & LIMITATIONS

The work prevents privacy violations of storage bits on a Standard Bloom using a simple binary DP mechanism. UltraFilter performs slightly less than the Standard Bloom filter but offers higher privacy guarantees, thereby trading utility for privacy. False negatives are not allowed in the Standard Bloom filter. On the contrary, UltraFilter has higher false negatives in connection with the noise level, ε, as it incurs some performance while increasing privacy levels. Finally, the Bloom filter is a widely used data structure for approximate set membership in diverse applications. Our evaluation is limited to testing UltraFilter on random text deviating from real-world settings where a uniform distribution may not suffice. Finally, following the proofs in Section 6, we have demonstrated the robustness of our solution to the privacy issues identified in Reviriego et al. [12].

REFERENCES

[1] Mohammad Alaggan, Sébastien Gambs, Stan Matwin, and Mohammed Tuhin. 2015. Sanitization of Call Detail Records via Differentially-Private Bloom Filters. In *Data and Applications Security and Privacy XXIX*. 223–230.

[2] Burton H. Bloom. 1970. Space/time trade-offs in hash coding with allowable errors. *Communication of the ACM* 13, 7 (1970), 422–426.

[3] Hanhua Chen, Liangyi Liao, Hai Jin, and Jie Wu. 2017. The Dynamic Cuckoo Filter. In *Proceedings of the International Conference on Network Protocols*. 1–10.

[4] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. 1995. Private Information Retrieval. In *Proceedings of the Symposium on Foundations of Computer Science*. 41.

[5] Adina Crainiceanu and Daniel Lemire. 2015. Bloofi: Multidimensional Bloom Filters. *Information Systems* 54, C (2015), 311–324.

[6] Jonathan L. Dautrich and Chinya V. Ravishankar. 2013. Inferential time-decaying Bloom filters. In *Proceedings of the International Conference on Extending Database Technology*. 239–250.

[7] Cynthia Dwork, Adam D. Smith, Thomas Steinke, and Jonathan Ullman. 2017. Exposed! A Survey of Attacks on Private Data. *Annual Review of Statistics and Its Application* 4 (2017), 61–84.

[8] Bin Fan, Dave G. Andersen, Michael Kaminsky, and Michael D. Mitzenmacher. 2014. Cuckoo Filter: Practically Better Than Bloom. In *Proceedings of the International on Conference on Emerging Networking Experiments and Technologies*. 75–88.

[9] Michael T. Goodrich and Michael Mitzenmacher. 2011. Invertible Bloom Lookup Tables. In *2011 49th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. 792–799.

[10] Tommi Meskanen, Jian Liu, Sara Ramezanian, and Valtteri Niemi. 2015. Private Membership Test for Bloom Filters. In *IEEE Trustcom/BigDataSE/ISPA*, Vol. 1. 515–522.

[11] Thomas G. Mueller and Daniel Lemire. 2019. Xor Filters: Faster and Smaller Than Bloom and Cuckoo Filters. *CoRR* abs/1912.08258 (2019). http://arxiv.org/abs/1912.08258

[12] Pedro Reviriego, Alfonso Sanchez-Macian, Stefan Walzer, Elena Merino-Gomez, Shanshan Liu, and Fabrizio Lombardi. 2023. On the Privacy of Counting Bloom Filters. *IEEE Transactions on Dependable and Secure Computing* 20, 2, 1488–1499.

6 APPENDIX

6.1 Tables

S/N	ε	$FP_{\hat{BF}}$	$FN_{\hat{BF}}$	$F1_{\hat{BF}}$
1	0	0.03	0.5	0.652
2	5	0.06	0.007	0.967
3	10	0.061	3.72e-5	0.97
4	15	0.061	0	0.97
5	20	0.061	0	0.97

Table 1: Noise, ε, on UltraFilter

S/N	φ	$F1_{BF}$	$F1^1_{\hat{BF}}$	$F1^2_{\hat{BF}}$	$F1^3_{\hat{BF}}$	$F1^4_{\hat{BF}}$
1	0.2	0.994	0.84	0.931	0.97	0.985
2	0.4	0.988	0.836	0.926	0.964	0.979
3	0.6	0.982	0.832	0.921	0.959	0.973
4	0.8	0.976	0.827	0.915	0.953	0.967
5	1.0	0.97	0.823	0.911	0.948	0.962

Table 2: Occupancy ratio, φ, over a range of noise, ε, on UltraFilter (Standard Bloom filter) (F1-score)

S/N	φ	FN_{BF}	$FN^1_{\hat{BF}}$	$FN^2_{\hat{BF}}$	$FN^3_{\hat{BF}}$	$FN^4_{\hat{BF}}$
1	0.2	0	0.269	0.12	0.047	0.018
2	0.4	0	0.269	0.119	0.0479	0.018
3	0.6	0	0.269	0.119	0.047	0.018
4	0.8	0	0.269	0.12	0.047	0.018
5	1.0	0	0.269	0.119	0.0475	0.018

Table 3: Occupancy ratio, φ, over a range of noise, ε, on UltraFilter(Standard Bloom filter) (FN-score)

S/N	φ	FP_{BF}	$FP^1_{\hat{BF}}$	$FP^2_{\hat{BF}}$	$FP^3_{\hat{BF}}$	$FP^4_{\hat{BF}}$
1	0.2	0.012	0.009	0.011	0.012	0.012
2	0.4	0.024	0.018	0.021	0.023	0.024
3	0.6	0.037	0.027	0.032	0.035	0.036
4	0.8	0.049	0.035	0.043	0.047	0.048
5	1.0	0.06	0.044	0.053	0.058	0.059

Table 4: Occupancy ratio, φ, over a range of noise, ε, on UltraFilter(Standard Bloom filter) (FP-score)

6.2 Algorithms

The underlying algorithms for UltraFilter in Algorithms 1, 2, and 3.

Algorithm 1 Insertion of Data

```
Require: data, $x$, bloom filter array, $BF$, and hash family, $H$
Ensure: $BF$
1: for $i \leftarrow 1$ to $l$ do
2:     $h_i \leftarrow H_i(x)$
3:     $BF[h_i] = 1$
4: end for
```

Algorithm 2 Lookup of Data

```
Require: data, $x$, Ultrafilter array, $\hat{BF}$, and hash family, $H$
Ensure: $ispresent$ Check if the element is present in a set
1: $ispresent \leftarrow True$
2: for $i \leftarrow 1$ to $l$ do
3:     $h_i \leftarrow H_i(x)$
4:     if $\hat{BF}[h_i] == 0$ then
5:         $ispresent \leftarrow False$
6:     end if
7: end for
```

[2] https://en.wikipedia.org/wiki/ASCII

[3] https://learn.microsoft.com/en-us/cpp/c-runtime-library/unicode-the-wide-character-set?view=msvc-170

Algorithm 3 Randomizer

```
Require: bloom filter array, $BF$, and noise level, $\varepsilon$
Ensure: $\hat{BF}$
1: Flip bit positions in $BF$ by setting to 1 with probability $\frac{e^{\varepsilon}}{e^{\varepsilon}+1}$.
2: Copy $BF$ into $\hat{BF}$.
```

Definition 6.1. The estimation of hamming distance between adjacent vectors is bounded by $|\mathcal{M}(BF(x)) - \mathcal{M}(\hat{BF}(x))| \leq \frac{e^{\varepsilon}}{e^{\varepsilon}+1}m$ where x is the input data with $(BF(x), \hat{BF}(x))$ showing the configurations of the storage bit array of the bloom filter.

Proof: We ensure that $\hat{BF}(x)$ is one of every possible neighbor bit array configuration after applying a DP transformation on $BF(x)$. $\hat{BF}(x)$ is only one transformation away from $BF(x)$. The neighborhood transformation is employed on the level of a bit array rather than on individual storage bits. Based on observation of the bits that are flipped at most during randomization where $\text{Ham}(BF(x), \hat{BF}(x))$ is the hamming distance between a pair of adjacent vectors, $(BF(x), \hat{BF}(x))$ and n is array length respectively. A case of adjacency relation between a pair of vectors($BF, \hat{BF}(x)$) where $\hat{BF}(x)$ is the randomized vector, $BF(x)$. An attacker aiming to reverse engineer the original vector after randomization would incur the reconstruction error, e where $e = \frac{\text{Ham}(BF(x), \hat{BF}(x))}{m}$ as shown in Equation 1. Tuning the noise, ε can impact the reconstruction error, influencing the achievable privacy levels.

$$\text{Ham}(BF(x), \hat{BF}(x)) \leq \frac{e^{\varepsilon}}{e^{\varepsilon}+1}m \tag{1}$$

Where $\text{Ham}(BF(x), \hat{BF}(x))$ is the hamming distance between a pair of adjacent vectors, $(BF(x), \hat{BF}(x))$ and m is array length.

Definition 6.2. The estimation of hamming distance between adjacent vectors is bounded by $|\mathcal{M}(BF(x)) - \mathcal{M}(\hat{BF}(y))| \leq m$ or $|\mathcal{M}(BF(x)) - \mathcal{M}(BF(y))| \leq m$ or $|\mathcal{M}(\hat{BF}(x)) - \mathcal{M}(\hat{BF}(y))| \leq m$ where every distinct data pair x, y is the input data with $(BF(x), \hat{BF}(y))$ showing the configurations of the storage bit array of the bloom filter.

Proof: By observation, for every distinct pair x, y, the hamming distance is upper bounded by the size of the bit array of the bloom filter as shown in Equation 2.

$$\text{Ham}(BF(x), \hat{BF}(y)) \leq m \tag{2}$$

Equation 2 holds similarly for $\text{Ham}(BF(x), \hat{BF}(y))$, $\text{Ham}(\hat{BF}(x), BF(y))$, or $\text{Ham}(\hat{BF}(x), \hat{BF}(y))$.

Claim 1: **Algorithm 3** is ε-differentially-private

Proof: Let us define the bloom filter array as BF vector before randomization and $\hat{BF}$ after randomization. The BF vector is randomized by flipping bits using the probability of random bits, B, defined in Equation 3.

$$\Pr(B=1) = \frac{e^{\varepsilon}}{e^{\varepsilon}+1}, \quad \Pr(B=0) = \frac{1}{e^{\varepsilon}+1} \tag{3}$$

For every pair of data as x, y with each pair of neighboring storage bitset array depicted as BF_1, BF_2 with $\text{Ham}(BF_1(x), BF_2(y)) \leq k$ where differential private mechanism, $\mathcal{M}$ is used to transform a bit set array (BF_1, BF_2).

$$\text{P}(\mathcal{M}(BF_1(x)) \in O) \leq e^{k\varepsilon}\text{P}(\mathcal{M}(BF_2(y)) \in O) + k\delta \tag{4}$$

Case 1: Following Equation 4, x and y are the same data. (BF_1 is BF and BF_2 is $\hat{BF}$) or (BF_2 is BF and BF_1 is $\hat{BF}$). Hence, the value of k is set to $\frac{e^{\varepsilon}}{e^{\varepsilon}+1}m$ which is a tighter bound following Definition 6.1. As a result, Equation 4 still holds.

Case 2: Following Equation 4, x and y are the different data. Every other case except ((BF_1 is BF and BF_2 is $\hat{BF}$) or (BF_2 is BF and BF_1 is $\hat{BF}$)) already covered in Case 1. We have considered the worst-case bound for the value of k set to m following Definition 6.2. As a result, Equation 4 still holds.

Deduction: Finally, fixing $\delta = 0$, then Equation 4 can be rewritten as

$$-k\varepsilon \leq \ln \frac{\mathcal{M}(\hat{BF}(x)))}{\mathcal{M}(BF(y))} \leq k\varepsilon$$

for every pair $x, y \in O$. Thus, Algorithm 3 is a differentially-private mechanism where $\mathcal{M}$ is a randomizer.

6.3 Lower Bound estimate of False Positive Rate, FP, and False Negative Rate, FN

Due to the probabilistic flipping of bits in UltraFilter, $FP_{\hat{BF}}$ has a higher value than FP_{BF} with the parameters set constant for both estimations as shown in Table 1. The lower bound estimates of False Positive Rate are presented in Equation 5 and Equation 6. Hence, for a standard bloom filter, false negatives are rare $FN_{BF} = 0$. Alternatively, UltraFilter permits false negatives, $FN_{\hat{BF}}$, as seen in Tables 1. Lower bound estimates of False Negative Rate exhibited in Equation 7 and Equation 8.

p is the flipping probability where

$$q = 1 - p, \qquad \varphi(p, m) = 1 - \frac{p}{m}$$

and the parameters defined as n is the number of records in bloom filter, l is the number of hash functions, and m is the size of bloom filter $\hat{BF}, BF$ as shown in Equations 5, 6, 7 and 8.

$$p = 1$$
$$FP_{BF} = \left(1 - \varphi(p, m)^{nl}\right)^{l} \tag{5}$$

$$p = \frac{e^{\varepsilon}}{e^{\varepsilon}+1}$$
$$FP_{\hat{BF}} = \left(1 - \varphi(p, m)^{nl}\right)^{l} \tag{6}$$

$$p = 1, q = 0$$
$$FN_{BF} = \left(1 - \varphi(q, m)^{nl}\right)^{l}$$
$$FN_{BF} = 0 \tag{7}$$

$$p = \frac{e^{\varepsilon}}{e^{\varepsilon}+1}$$
$$q = \frac{1}{e^{\varepsilon}+1}$$
$$FN_{\hat{BF}} = \left(1 - \varphi(q, m)^{nl}\right)^{l} \tag{8}$$