Adversary Resilient Learned Bloom Filters

Allison Bishop[1,2][0000−0003−3986−8985] and Hayder Tirmazi[1][0009−0008−9360−9662]

[1] City College of New York
[2] Proof Trading
abishop@ccny.cuny.edu, hayder.research@gmail.com

Abstract. The Learned Bloom Filter is a recently proposed data structure that combines the Bloom Filter with a Learning Model while preserving the Bloom Filter's one-sided error guarantees. Creating an adversary-resilient construction of the Learned Bloom Filter with provable guarantees is an open problem. We define a strong adversarial model for the Learned Bloom Filter. Our adversarial model extends an existing adversarial model designed for the Classical (i.e. not "Learned") Bloom Filter by prior work and considers computationally bounded adversaries that run in probabilistic polynomial time (PPT). Using our model, we construct an adversary-resilient variant of the Learned Bloom Filter called the Downtown Bodega Filter. We show that: if pseudo-random permutations exist, then an Adversary Resilient Learned Bloom Filter may be constructed with 2λ extra bits of memory and at most one extra pseudo-random permutation in the critical path. We construct a hybrid adversarial model for the case where a fraction of the query workload is chosen by an adversary. We show realistic scenarios where using the Downtown Bodega Filter gives better performance guarantees compared to alternative approaches in this hybrid model.

Keywords: Secret Key Cryptography · Adversarial Artificial Intelligence · Probabilistic Data Structures.

1 Introduction

The Bloom Filter is a probabilistic data structure that solves the Approximate Membership Query Problem. The data structure now known as the "Bloom" Filter was initially proposed as method 2 in the section "Two Hash-Coding Methods with Allowable Errors" in a 1970 paper by Burton H. Bloom [1, 2]. The Bloom Filter has applications in databases, cryptography, computer networking, social networking [3], and network security [4]. The Learned Bloom Filter is a novel data structure first proposed by Kraska et al [5] in 2017. The Learned Bloom Filter can be thought of as a Bloom Filter working in collaboration with a Learning Model. There are currently no known provably secure constructions of the Learned Bloom Filter. Prior work [6] has left the security of the Learned Bloom Filter as an open problem.

In this section, we first summarize the original formulation of the Bloom Filter and then provide background on the Learned Bloom Filter. Next, we outline the scope of this work and provide a summary of our results. Lastly, we motivate our results and discuss related work.

1.1 The Bloom Filter

A Bloom Filter representing a set S may have false positives ($s \notin S$ may return true) but does not have false negatives ($s \in S$ is always true). The Learned Bloom Filter provides the same correctness guarantees as the Bloom Filter but with potentially better performance for the same memory budget. We first discuss the (Classical) Bloom Filter and then discuss the Learned Bloom Filter.

Classical Bloom Filter Figure 1 provides a helpful illustration of a Bloom Filter and the insert and check operations we introduce below.

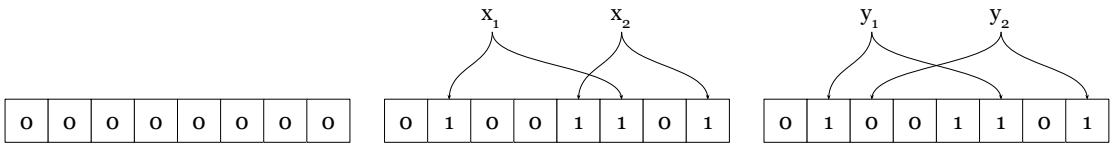


Fig. 1: Example of a Bloom Filter with $m = 8$ and $k = 2$. Initially, all m bits are unset. Each element x_i is hashed k times, and each corresponding bit is set. To check each element y_i, the element is hashed k times. If any corresponding bit is unset, the element y_i is not in set S (with probability 1). If all corresponding bits are set, the element y_i is either in set S or the element y_i has caused the Bloom Filter to return a false positive

Definition 1 (Bloom Filter). *A Bloom Filter for representing set S with cardinality n is a zero-initialized array of m bits. A Bloom Filter requires k independent hash functions h_i such that the range of each h_i is the set of integers $\{1, \ldots, m\}$ [4].*

Most mathematical treatments such as Mitzenmacher and Broder [4] make the convenient assumption that each h_i maps each item in the universe to a random number uniformly over the (integer) range $[1, m]$. In the remainder of this text, we shall refer to this formulation of the Bloom Filter as the **Classical** Bloom Filter. This is to distinguish it from the **Learned** Bloom Filter, which our results are focused on.

Definition 2 (Insert Operation). *For each element $x \in S$, the bits $h_i(x)$ are set to 1 for $i \in [1, k]$.*

If a bit already set to 1 is set to 1 again, its value remains 1 i.e. a double set does not flip the bit back to 0.

Definition 3 (Check Operation). *For an element x, we return true if all $h_i(x)$ map to bits that are set to 1. If there exists an $h_i(x)$ that maps to a bit that is 0, we return false.*

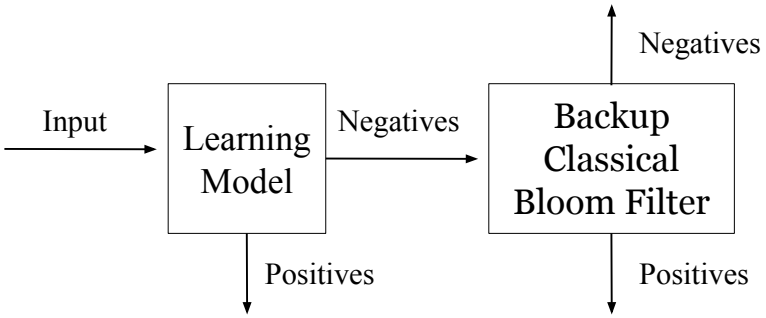


Fig. 2: Example of a Learned Bloom filter with a Learning Model and a Backup Classical Bloom filter that only checks values that are with high probability negative in the Bloom Filter Learning Model, to ensure a one-sided error bound (i.e. only false positives and no false negatives).

Learned Bloom Filter The **Learned** Bloom Filter is a novel data structure proposed by Kraska et al [5] in 2017. Kraska et al. suggest using a pre-filter ahead of the Classical Bloom Filter, where the pre-filter is derived from a learning model. The learning model estimates the probability of an element x being in the set S. This allows the use of a smaller (in terms of memory) Classical Bloom Filter compared to the case where a Classical Bloom Filter is used alone. A mathematical model and guarantees for the Learned Bloom Filter were first provided by Mitzenmacher [7] in 2018. While Kraska et al. used a neural network for the pre-filter, Mitzenmacher generalizes the pre-filter to use any approach that estimates the probability of an element x being in the set S. We use this generalized formulation of the Learned Bloom Filter in this work. We introduce the Learned Bloom Filter using a self-contained definition adapted from Mitzenmacher [7]. We extend this definition in Section 2.2.

Definition 4 (Learned Bloom Filter). *A Learned Bloom filter on a set of positive keys $\mathcal{K}$ and negative keys $\mathcal{U}$ is a function $f : U \mapsto [0, 1]$ and threshold τ, where U is the universe of possible query keys, and an associated Classical Bloom Filter B is referred to as a Backup Classical Bloom filter. The Backup Classical Bloom Filter holds the set of keys $\{z : z \in \mathcal{K}, f(z) < \tau\}$. For a query y, the Learned Bloom filter returns that $y \in \mathcal{K}$ if $f(y) \geq \tau$, or if $f(y) < \tau$ and the Backup Classical Bloom Filter returns that $y \in \mathcal{K}$. The Learned Bloom filter returns $y \notin \mathcal{K}$ otherwise.*

The Learned Bloom Filter provides better performance on the false positive rate while maintaining the guarantee of having no false negatives. We show an example of a Learned Bloom Filter adapted from Mitzenmacher et al [7] in figure 2.

1.2 Our Results

We introduce an adversarial model for the Learned Bloom Filter. Our model extends the adversarial model for the Classical Bloom Filter introduced by Naor and Yogev [8]. Using our adversarial model, we introduce the notion of an *adversarial resilient* Learned Bloom Filter, as well as an *adversarial* ***reveal*** *resilient* Learned Bloom Filter. Informally, an adversarial reveal resilient Learned Bloom Filter is difficult for an adversary to generate false positives for, even if the adversary knows the internal representation of the set encoded by the Bloom Filter. We also describe the Sandwiched Learned Bloom Filter, a well-known variant of the Learned Bloom Filter consisting of a Learning Model "sandwiched" between two Classical Bloom Filters. The precise definitions and the adversarial model we introduce are provided in Section 2.

Secure Learned Bloom Filter We introduce the first provably secure constructions of the Learned Bloom Filter and show that using pseudo-random permutations, one can create provably secure constructions of the Learned Bloom Filter with at most 2λ extra bits of memory, where λ is the security parameter.

Theorem 1 (Secure Constructions). *Let B be an (n, ϵ)-Sandwiched Learned Bloom Filter using m bits of memory. If pseudo-random permutations exist, we show that there exists a negligible function* $\mathrm{negl}(\cdot)$ *such that for security parameter λ there exists an $(n, \epsilon + \mathrm{negl}(\lambda))$-adversarial* ***reveal*** *resilient Learned Bloom Filter that uses $m' = m + 2\lambda$ bits of memory*

The parameters n and ϵ are a lower bound on the security parameter λ and an upper bound on the false positive probability respectively. They are precisely defined in Section 2. This result is formally proved in Theorem 7 of Section 3.

Utility of the Secure Learned Bloom Filter To explore the utility of the Secure Learned Bloom Filter, we introduce a hybrid model where an adversary is allowed to choose αN queries out of a workload of N queries sent to the Bloom Filter. The utility of the Bloom Filter is commonly measured by looking at its False Positive Rate (FPR) in comparison to its memory usage.

Theorem 2 (Performance in the Hybrid Model). *For a given memory budget $M = m_L + m_A + m_B + 2\lambda$, any set S and corresponding training dataset $\mathcal{D}$, in the hybrid adversarial setting with given $\alpha = \alpha_P + \alpha_N$, the expected false positive probability of a Secure Learned Bloom Filter is lower than the expected false positive probability of a Secure Classical Bloom Filter if the following holds:*

$$\begin{aligned} \alpha_P FPR(S'', m_A) + \alpha_N FPR(S', m_B) + (1 - \alpha_P - \alpha_N) FPR_{DB}(S, \mathcal{D}, M) \\ < FPR(S, m_L + m_A + m_B + \lambda) \end{aligned} \tag{1}$$

where $S' \subset S$ is the set of elements in S for which the Bloom Filter Learning Model L returns negative, $S'' \subset S$ is the set of elements in S for which the Bloom Filter Learning Model L returns positive, where the FPR is a function for the false positive probability of a Secure Classical Bloom Filter, and FPR_{DB} is the function for the false positive probability of a Secure Learned Bloom Filter.

A training dataset is a collection of positive and negative query results used to train the Learning Model. By a hybrid adversarial setting, we mean an adversarial model where a fraction of the queries are adversary-generated. We precisely define these notions in Section 2.2 and Section 5 respectively. These results are formally proven in Theorem 10 of Section 5. In this result, when we refer to a "Secure Learned Bloom Filter", we refer in particular to the "Downtown Bodega Filter" construction we define in Section 3.2. We demonstrate multiple settings in a hybrid adversarial model where the Secure Learned Bloom Filter outperforms the Secure Classical Bloom Filter (Section 5.4). The advantage of the Secure Learned Bloom Filter compared to the Secure Classical Bloom Filter intricately depends on the number of adversarial queries and the performance of the Bloom Filter Learning Model.

1.3 Motivation

The Bloom Filter and its variants have numerous applications in computing [9, 4]. We borrow discussion on Bloom Filter applications from the survey by Tarkoma et al [9]. The Bloom Filter may be implemented in kernel space in a Linux network driver for performant filtering of network packets. Loop detection in network protocols and multicast forwarding engines may also utilize the Bloom Filter. Deep Packet Scanners and Packet Classifiers have also found the Bloom Filter helpful for improving efficiency. The Bloom Filter may be used to detect heavy flows in network traffic from the vantage point of a router. The Bloom Filter has also been used in the OPUS system [9] that stores a list of words that involve poor password choices encouraging users to select better passwords. The Bloom Filter has also found success in the detection of hash tampering in network-attached disks. Google's BigTable system uses the Bloom Filter to minimize disk reads. Apache Hadoop also uses the Bloom Filter as an optimization in the reduce stage of its map/reduce implementation. Other applications of the Bloom Filter include uses in the realms of peer-to-peer networking and caching.

A large number of the applications of the Bloom Filter involve critical infrastructure [10]. It is possible to forge false positives in a naively implemented Bloom Filter [10] allowing an adversary to make the Bloom Filter deviate from its behavior. Gerbet et al [10] show practical attacks on the

Scrapy web-spider, the Bitly Dablooms spam filter, and the Squid web cache. Naor and Yogev [8] motivate the need for securing the Bloom Filter by considering a white list of email addresses for spam filtering. In their scenario, an adversary that can forge false positives may easily infiltrate the spam filter.

1.4 Related Work

Section 1.4 provides a thorough overview of prior work on adversarial models and the security of the Classical Bloom Filter. Similarly, Section 1.4 discusses prior work on the security of the Learned Bloom Filter.

Classical Bloom Filter Gerbet et al. [10] suggest practical attacks on the Classical Bloom Filter and the use of universal hash functions and message authentication codes (MACs) to mitigate a subset of those attacks. Naor and Yogev [8] define an adversarial model for the Classical Bloom Filter and use it to prove that (1) for computationally bounded adversaries, non-trivial adversary resilient Bloom filters exist if and only if one-way functions exist, and (2) for computationally unbounded adversaries, there exists a Classical Bloom Filter that is secure against t queries while using only $\mathcal{O}(n \log \frac{1}{\epsilon} + t)$ bits of memory. n is the size of the set and ϵ is the desired error. We borrow their idea of using Pseudorandom Permutations for the Classical Bloom Filter and apply it to the Learned Bloom Filter.

There are multiple adversary models in literature for the Classical Bloom Filter. Naor and Oved [11] unify this line of research by presenting several robustness notions in a generalized adversarial model for the Classical Bloom Filter. Naor and Oved [11] give the security notion of Naor and Yogev [8] the name *Always-Bet (AB)* test. They extend this and present a new, strictly stronger, security notion called the *Bet-Or-Pass (BP)*.

Multiple Probablistic Data Structures Clayton et al. [12] analyze the Classical Bloom Filter, the Counting Bloom Filter, and the Count-Min Sketch in an adversarial setting. Clayton et al. use a stronger adversarial model than Naor and Yogev [8], allowing an adversary to perform insertions and giving an adversary access to the internal state of the Classical Bloom Filter. Clayton et al. propose using salts and keyed pseudo-random functions for securing the Classical Bloom Filter. They do not address Learned Probabilistic Data Structures including the Learned Bloom Filter. Both Naor and Yogev [8], and Clayton et al. [12], perform their analysis in a game-based setting.

Filic et al. [13] investigate the adversarial correctness and privacy of the Classical Bloom Filter and an insertion-only variant of the Cuckoo Filter. Filic et al. also use a stronger adversarial model than Naor and Yogev [8] allowing an adversary to insert entries into the Classical Bloom Filter and query for the internal state of the Classical Bloom Filter. Filic et al. [13] perform their analysis in a simulator-based setting. None of the works discussed above address *Learned* Probabilistic Data Structures including the Learned Bloom Filter.

Learned Bloom Filter The authors are only aware of one prior work that addresses the Learned Bloom Filter in an adversarial setting, Reviriego et al [6]. They propose a practical attack on the Learned Bloom Filter. They suggest two possible mitigations for their proposed attack: swapping to a Classical Bloom Filter upon detection of the attack or adding a second Backup Classical Bloom Filter. However, they do not provide any provable guarantees on the performance of the Learned

Bloom Filter in the presence of adversaries. They leave the security of the Learned Bloom Filter as an open problem in their work.

2 Adversarial Model

We first describe the adversarial model of Naor and Yogev [8] for the Classical Bloom Filter and then use it as the basis for creating an adversarial model for the Learned Bloom Filter. We refer to the adversarial model defined by Naor and Yogev [8] as the *classical* adversarial model. Section 2.1 contains a treatment of the classical adversarial model. We also introduce a stronger adversary than the one described in Naor and Yogev 's [8] model that has access to the internal state of the Classical Bloom Filter. Section 2.2 introduces a definition of the Learned Bloom Filter adapted from [7] and discusses extensions to the classical adversarial model to make it work with the Learned Bloom Filter.

2.1 Classical Adversarial Model

Let S be a finite set of cardinality n in a suitable finite universe U of cardinality u. Let M be a compressed representation of S. Let r be any random string and M_r^S be a compressed representation of S with r. Let λ be a security parameter. Let $A = (A_C, A_Q)$ be any probabilistic polynomial time (PPT) adversary.

Definition 5 (Construction). *We define C to be a setup algorithm such that $C(1^\lambda, S) = M$. We define C_r, the randomized version of C, to be a setup algorithm such that $C_r(1^\lambda, S) = M_r^S$. Note that M and M_r^S are both compressed representations of S, as defined above.*

As a running example, let x be an element and consider the set $S = \{x\}$ in a Classical Bloom Filter that uses 2 hash functions, h_1, h_2 and 4 bits such that $h_1(x) = 1$ and $h_2(x) = 3$. A trivial deterministic setup algorithm, on input S, would then generate the representation $M_r^S = 1010$. The first and third bits in this representation are set according to the example.

Definition 6 (Query). *We are provided a set S and a compressed representation of that set, M_r^S (where r is any random string). We define Q_s to be a query algorithm such that $Q_s(M_r^S, x) = 1$ if $x \in S$, and $Q_s(M_r^S, x) \in \{0, 1\}$ if $x \notin S$. Q_s must not be randomized and must not change M_r^S.*

In our running example, a trivial query algorithm returns 1 if and only if all hashes for an element return indexes that are set i.e $Q_s(M_r^S, x) = (M_r^S(h_1(x)) = 1 \wedge M_r^S[h_2(x)] = 1)$. Consider a new element y for which $h_1(y) = 1$ and $h_2(y) = 2$. With $M_r^S = 1010$, $Q(M_r^S, x)$ returns 1 since both indices 1 and 3 are set, however $Q(M_r^S, y)$ return 0 as index 2 is not set.

Definition 7. *We define Q_u to be a query algorithm similar to Q_s differing only in that Q_u may be randomized and it may change the compressed representation of S, M_r^S, after each query.*

We now give a precise definition for the Classical Bloom Filter in an adversarial setting.

Definition 8 (Classical Bloom Filter). *Let a Classical Bloom Filter be a data structure $B = (C_r, Q)$ where C_r obeys Definition 5 and Q obeys either Definition 6 or Definition 7.*

We define a special class of Classical Bloom Filters which were coined "steady" Classical Bloom Filters by Naor and Yogev [8]. Steady Classical Bloom Filters do not change their internal representation M_r^S after the setup algorithm C_r has been executed. In other words, only query algorithms of the type Q_s are permitted in the steady setting, and query algorithms of the type Q_u are not permitted.

Definition 9 (Steady). *Let a steady (n, ϵ)-Classical Bloom Filter be a Classical Bloom Filter $B_s = (C_r, Q_s)$ such that Q_s obeys Definition 6 and $\forall x \in U$, it holds that*

1. *Completeness: $\forall x \in S : P[Q_s(C_r(S), x) = 1] = 1$*
2. *Soundness: $\forall x \notin S : P[Q_s(C_r(S), x) = 1] \leq \epsilon$*

where the probabilities are taken over C_r.

Referring again to our running example, our trivial query algorithm is complete as for element x which already exists in S, both index 1 ($h_1(x) = 1$) and index 3 ($h_2(x) = 3$) are set, so our trivial query algorithm returns 1. Our trivial query algorithm is also sound because for any $y \neq x$, $P[h_1(y) \in \{1, 3\}]$ is bounded and $P[h_2(y) \in \{1, 3\}]$ is bounded, therefore the probability of our trivial query algorithm returning 1 is bounded.

Now we construct our first adversarial challenge for the Classical Bloom Filter in the steady setting. A probabilistic polynomial time (PPT) adversary A, as defined at the start of this section, is given a security parameter $1^{\lambda+n\log(u)}$ and is allowed to construct a set S. The set S is then given to construction algorithm C_r along with the security parameter to yield representation M_r^S. The adversary is allowed t queries to the query algorithm Q_s for which it is provided the results. After the t queries, the adversary must output an element x^*. If x^* is a false positive and has not been queried before, the adversary wins the challenge. Otherwise, the adversary loses the challenge. We define this precisely in Challenge 3

Challenge 3 (Resilient) *We denote this challenge as ${\Lambda^1}_{A,t}(\lambda)$.*

1. *$S \leftarrow A_C(1^{\lambda+n\log(u)})$*
2. *$M_r^S \leftarrow C_r(1^{\lambda+n\log(u)}, S)$*
3. *$x^* \leftarrow A_Q^{Q_s(M_r^S,\cdot)}(1^{\lambda+n\log(u)}, S)$. A_Q performs at most t queries $x_1, \ldots, x_t$ to $Q_s(M_r^S, \cdot)$.*
4. *If $x^* \notin S \cup \{x_1, \ldots, x_t\}$ and $Q_s(M_r^S, x^*) = 1$, output 1. Otherwise, output 0.*

We now define an adversarial resilient Classical Bloom Filter based on the random variable ${\Lambda^1}_{A,t}(\lambda)$.

Definition 10 (Resilient).
Let an (n, t, ϵ)-adversarial resilient steady Classical Bloom Filter be any steady Classical Bloom Filter for which it holds that, $\forall \lambda > n \in \mathbb{N}, P[\Lambda^1_{A,t}(\lambda) = 1] \leq \epsilon$.

Now, we create an extension to Naor and Yogev 's [8] model, introducing a stronger adversary that has access to the internal state of the Classical Bloom Filter. We construct our second adversarial challenge for the Classical Bloom Filter in the steady setting. This challenge is almost identical to the first challenge with the only difference being that the adversary is allowed access to the representation M_r^S. We define our second challenge more precisely in Challenge 4.

Challenge 4 (Reveal Resilient) *We denote this challenge as ${\Lambda^2}_{A,t}(\lambda)$.*

1. $S \leftarrow A_C(1^{\lambda+n\log(u)})$
2. $M_r^S \leftarrow C_r(1^{\lambda+n\log(u)}, S)$
3. $x^* \leftarrow A_Q^{Q_s(M_r^S,\cdot)}(1^{\lambda+n\log(u)}, S, M_r^S)$. *$A_Q$ performs at most t queries $x_1, \ldots, x_t$ to $Q_s(M_r^S, \cdot)$.*
4. *If $x^* \notin S \cup \{x_1, \ldots, x_t\}$ and $Q_s(M_r^S, x^*) = 1$, output 1. Otherwise, output 0.*

Analogous to Definition 10, we now define an adversarial reveal resilient Classical Bloom Filter based on the random variable $\Lambda_{A,t}^2(\lambda)$ (from Challenge 4).

Definition 11 (Reveal Resilient).
*Let an (n, t, ϵ)-adversarial **reveal** resilient steady Classical Bloom Filter be any steady Classical Bloom Filter for which it holds that, $\forall \lambda > n \in \mathbb{N}, P[\Lambda_{A,t}^2(\lambda) = 1] \leq \epsilon$.*

2.2 Learned Adversarial Model

In this section, we first discuss a mathematical model for the Learned Bloom Filter. We then create an adversarial model based on the Learned Bloom Filter. We define challenges and security definitions for the Learned Bloom Filter that are analogous to the ones we defined for the Classical Bloom Filter in Section 2.1.

Mitzenmacher [7] was the first to create a mathematical model for the Learned Bloom Filter, in 2018. Our model is heavily based on Mitzenmacher's model but with some additional definitions to suit our adversarial setting. Consider a set of elements $\mathcal{K} \subset S$ and a set of elements $\mathcal{U}$ such that $\forall u \in \mathcal{U}, u \notin S$. We form a training dataset $\mathcal{D} = \{(x_i, y_i = 1) | x_i \in \mathcal{K}\} \cup \{(x_i, y_i = 0) | x_i \in \mathcal{U}\}$.

Definition 12 (Dataset Construction). *Let Δ_r be any construction algorithm that takes a set S, and constructs a training dataset $\mathcal{D}$ for S.*

Definition 13 (Learning Model). *Let a Bloom Filter Learning Model, $l : U \mapsto [0, 1]$, be any function that maps elements in a suitable finite universe to a probability.*

We train a Bloom Filter Learning Model, l_r, on $\mathcal{D}$. Let $l_r(x)$ be the probability estimate from the learning model that x is an element in S. A value τ may be chosen as a threshold. When $l(x) \geq \tau$ then the Learned Bloom Filter considers x to be an element of S. Otherwise, the Learned Bloom Filter passes x onto the Backup Classical Bloom Filter. Figure 2 provides a helpful illustration.

Definition 14 (Learned Construction). *We define $\tilde{C}_r$ to be a setup algorithm such that $\tilde{C}_r(1^\lambda, S, \mathcal{D}) = \tilde{M}_r^S$, where $\tilde{M}_r^S$ is a learned compressed representation of S i.e a compressed representation that includes a Bloom Filter Learning Model.*

Returning to our running example, consider the set $S = \{x, y\}$ and the dataset $\mathcal{D} = \{(x, 1), (y, 1), (z, 0)\}$ in a Learned Bloom Filter that uses the Bloom Filter Learning Model $l_r^{\mathcal{D}}$ such that $l_r^{\mathcal{D}}(x) = 0.6$ and $l_r^{\mathcal{D}}(y) = 0.4$, and uses the threshold $\tau = 0.5$. Let $M_r^{S'}$ be the compressed representation of the set $S' = \{x : x \in S | l(x) < \tau\} = \{y\}$ created by any setup algorithm for the **Classical** Bloom Filter. A trivial setup algorithm for the **Learned** Bloom Filter will then return the compressed representation $\tilde{M}_r^S = (l(x), \tau, M_r^{S'})$.

Definition 15 (Learned Query). *We define $\tilde{Q}_s$ to be a query algorithm similar to Q_s (Definition 6) differing only in that $\tilde{Q}_s$ only takes a learned compressed representation $\tilde{M}_r^S$ of the set S instead of any compressed representation M_r^S.*

In our running example, a trivial query algorithm, $\tilde{Q}_s(\tilde{M}_r^S, x)$ would then be $l(x) \geq 0.5 \vee (M_r^{S'}[h_1(x)] = 1 \wedge M_r^{S'}[h_2(x)] = 1)$.

Definition 16 (Learned Bloom Filter). *Let a Learned Bloom Filter be a data structure $\tilde{B} = (\tilde{C}_r, \tilde{Q}_s)$ where $\tilde{C}_r$ obeys Definition 14 and $\tilde{Q}_s$ obeys Definition 15. The query algorithm $\tilde{Q}_s$ for the Learned Bloom Filter is*

$$\tilde{Q}_s = l(x) \geq \tau \vee Q_s(M_r^{S'}, x) = 1$$

where Q_s is a query algorithm for the Classical Bloom Filter, and $M_r^{S'}$ is the internal representation of the Backup Classical Bloom Filter encoding the set $S' = \{x \in S | l_r(x) < \tau\}$

It is trivial to define a steady Learned Bloom Filter with completeness and soundness properties analogous to the ones outlined in Definition 9 for a Classical Bloom Filter. We now construct an adversarial model for the Learned Bloom Filter.

Challenge 5 (Learned Resilient) *We denote this challenge as ${\Lambda^{1l}}_{A,t}(\lambda)$.*

1. $S \leftarrow A_C(1^{\lambda + n\log(u)})$
2. $\mathcal{D} \leftarrow \Delta_r(S)$
3. $\tilde{M}_r^S \leftarrow \tilde{C}_r(1^{\lambda+n\log(u)}, S, \mathcal{D})$
4. $x^* \leftarrow A_Q^{\tilde{Q}_s(\tilde{M}_r^S, \cdot)}(1^{\lambda+n\log(u)}, S)$. *$A_Q$ performs at most t queries $x_1, \ldots, x_t$ to $\tilde{Q}_s(\tilde{M}_r^S, \cdot)$.*
5. *If $x^* \notin S \cup \{x_1, \ldots, x_t\}$ and $\tilde{Q}_s(M_r^S, x^*) = 1$, output 1. Otherwise, output 0.*

Note that the adversary must not choose threshold τ. If the adversary is allowed to choose threshold τ, then Challenge 5 is easily succeeded by choosing $\tau = 0$. We are now ready to formally define security for the steady Learned Bloom Filter.

Definition 17 (Learned Resilient).
Let an (n, t, ϵ)-adversarial resilient steady Learned Bloom Filter be any steady Learned Bloom Filter for which it holds that, $\forall \lambda > n \in \mathbb{N}, P[\Lambda_{A,t}^{1l}(\lambda) = 1] \leq \epsilon$.

We now propose a stronger adversary that has access to the internal state of the Learned Bloom Filter.

Challenge 6 (Learned Reveal Resilient) *We denote this challenge as ${\Lambda^{2l}}_{A,t}(\lambda)$.*

1. $S \leftarrow A_C(1^{\lambda + n\log(u)})$
2. $\mathcal{D} \leftarrow \Delta_r(S)$
3. $\tilde{M}_r^S \leftarrow \tilde{C}_r(1^{\lambda+n\log(u)}, S, \mathcal{D})$
4. $x^* \leftarrow A_Q^{\tilde{Q}_s(\tilde{M}_r^S, \cdot)}(1^{\lambda+n\log(u)}, S, \tilde{M}_r^S)$. *$A_Q$ performs at most t queries $x_1, \ldots, x_t$ to $\tilde{Q}_s(\tilde{M}_r^S, \cdot)$.*
5. *If $x^* \notin S \cup \{x_1, \ldots, x_t\}$ and $\tilde{Q}_s(M_r^S, x^*) = 1$, output 1. Otherwise, output 0.*

Definition 18 (Learned Reveal Resilient).
*Let an (n, t, ϵ)-adversarial **reveal** resilient steady Learned Bloom Filter be any steady Learned Bloom Filter for which it holds that, $\forall \lambda > n \in \mathbb{N}, P[\Lambda_{A,t}^{2l}(\lambda) = 1] \leq \epsilon$.*

3 Secure Constructions for the Learned Bloom Filter

In this section, we propose a solution to the problem of securing the Learned Bloom Filter. We call our construction the Downtown Bodega Filter. Section 3.1 contains the background necessary to construct the Downtown Bodega Filter. We formally define the Downtown Bodega Filter in Section 3.2.

3.1 Preliminaries

Setting Review: Consider a set of elements $\mathcal{K} \subset S$ and a set of elements $\mathcal{U}$ such that $\forall u \in \mathcal{U}, u \notin S$. We form a dataset $\mathcal{D} = \{(x_i, y_i = 1) | x_i \in \mathcal{K}\} \cup \{(x_i, y_i = 0) | x_i \in \mathcal{U}\}$. We train a Bloom Filter Learning Model, $l_r^{\mathcal{D}}$, on the training dataset $\mathcal{D}$. Let $l_r^{\mathcal{D}}(x)$ be the probability estimate from the learning model (Definition 13) that x is an element in S. A value τ may be chosen as a threshold. When $l_r^{\mathcal{D}}(x) \geq \tau$ then the Learned Bloom Filter considers x to be an element of S. Otherwise, the Learned Bloom Filter passes x onto the Backup Classical Bloom Filter.

We introduce a Learned Bloom Filter technique called "sandwiching" introduced by Mitzenmacher [7]. We then discuss the use of a pseudo-random permutation on the Bloom Filter input set first proposed by Naor and Yogev [8].

Definition 19 (Sandwiched Learned). *Let a Sandwiched Learned Bloom Filter, $SB_r = (\tilde{C}_r, \tilde{Q}_s)$ be a data structure where $\tilde{C}_r$ obeys Definition 14, and $\tilde{Q}_s$ obeys Definition 15.*

The learned compressed representation of any set S and training dataset $\mathcal{D}$ under a Sandwiched Learned Bloom Filter consists of the following:

1. *A Bloom Filter Learning Model $l_r^{\mathcal{D}}$ trained on $\mathcal{D}$*
2. *A suitable threshold τ for $l_r^{\mathcal{D}}$*
3. *M_r^S, the compressed representation of the complete set S encoded by a Classical Bloom Filter. We refer to this Classical Bloom Filter as the Initial Classical Bloom Filter.*
4. *$M_r^{S'}$, the compressed representation of the set $S' = \{x : x \in S | l_r^{\mathcal{D}}(x) < \tau\}$ encoded by a Classical Bloom Filter. We refer to this Classical Bloom Filter as the Backup Classical Bloom Filter.*

The query algorithm $\tilde{Q}_s$ for the Sandwiched Learned Bloom Filter is

$$(Q_s(M_r^S, x) = 1) \wedge (l_r^{\mathcal{D}}(x) > \tau \vee Q_s(M_r^{S'}, x) = 1)$$

Where Q_s is a query algorithm for the Classical Bloom Filter.

Figure 3 shows an example of a Sandwiched Learned Bloom Filter. In our running example, the setup algorithm for a Sandwiched Learned Bloom Filter creates the same Bloom Filter Learning Model, l_r, as well as the same compressed representation for the Backup Classical Bloom Filter $M_r^{S'}$ (trained on the set $S' = \{y\}$) that we discussed after Definition 14. In addition, it also creates a representation M_r^S for the Initial Classical Bloom Filter trained on the complete set $S = \{x, y\}$. A trivial query algorithm $\tilde{Q}_s(\tilde{M}_r^S, x)$ for the Sandwiched Learned Bloom Filter would then be $(M_r^S[h_1(x)] = 1 \wedge M_r^S[h_2(x)] = 1) \wedge (l_r(x) > \tau \vee (M_r^{S'}[h_1(x)] = 1 \wedge M_r^{S'}[h_2(x)] = 1)$.

Lemma 1. *If x is a false positive in a Sandwiched Learned Bloom Filter, SB_r, then x is a false positive in the Initial Classical Bloom Filter of the Sandwiched Learned Bloom Filter.*

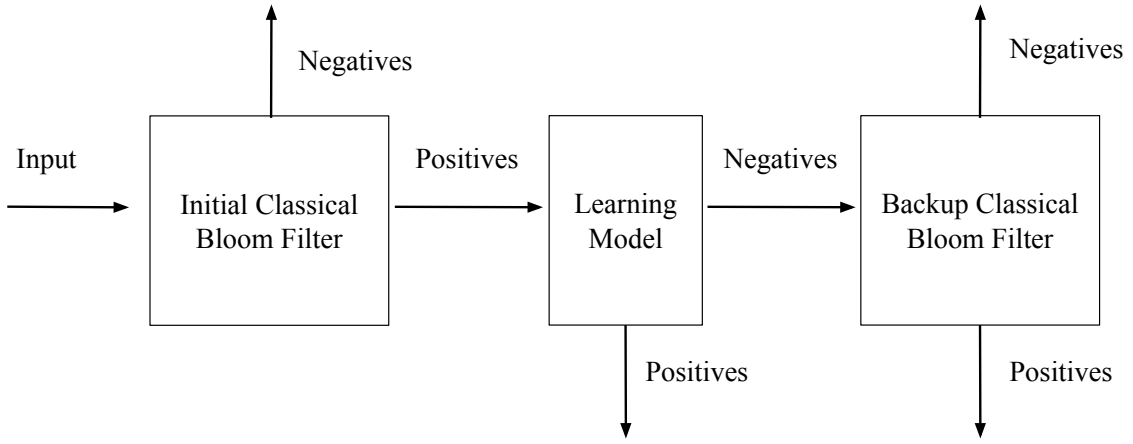


Fig. 3: A Sandwiched Learned Bloom Filter. The initial filter only allows positives (true positive and false positive) to reach the Learned Bloom Filter.

Proof. The proof follows from the definition of a Sandwiched Learned Bloom Filter. x is a false positive in one of two cases:

Case 1: The Learning Model returns positive on x. In this case, x only reaches the Learning Model if it was **not** marked negative by the Initial Classical Bloom Filter. Therefore it was a false positive in the Initial Classical Bloom Filter.

Case 2: The Backup Classical Bloom Filter returns positive on x. Using similar reasoning as Case 1, we can show that this only occurs if x was a false positive in the Initial Classical Bloom Filter.

We use the standard definitions for pseudo-random permutations in this work. We provide a brief but self-contained treatment of pseudo-random permutations adapted from Chapter 3 of Katz Lindell [14] in Appendix A. Theorem 4.8 of Naor and Yogev [8] proves that for a classical steady (n, ϵ)-Bloom Filter that uses m bits of memory, if pseudo-random permutations exist, then there exists a negligible function $\texttt{negl}$ such that for security parameter λ there exists a $(n, \epsilon + \texttt{negl}(\lambda))$-adversarial resilient Classical Bloom Filter that uses $m^{'} = m+\lambda$ bits of memory. This secure Classical Bloom Filter can be constructed by running the initialization algorithm on $S^{'} = \{F_k(x) : x \in S\}$ instead of S [8].

3.2 The Downtown Bodega Filter

We introduce a secure construction of the Learned Bloom Filter which we call the Downtown Bodega Filter.

Definition 20 (Downtown). *Let a Downtown Bodega Filter be a data structure* $DB_{r,k_A,k_B} = (F_{k_A}, F_{k_B}, \tilde{C}_r, \tilde{Q}_s)$ *where* $\tilde{C}_r$ *obeys Definition 14,* $\tilde{Q}_s$ *obeys Definition 15, and* F_{k_A}, F_{k_B} *are pseudo-random permutations. In other words,* $(\tilde{C}_r, \tilde{Q}_s)$ *form a Learned Bloom Filter.*

The learned compressed representation of any set S *and training dataset* $\mathcal{D}$ *under a Downtown Bodega Filter consists of the following:*

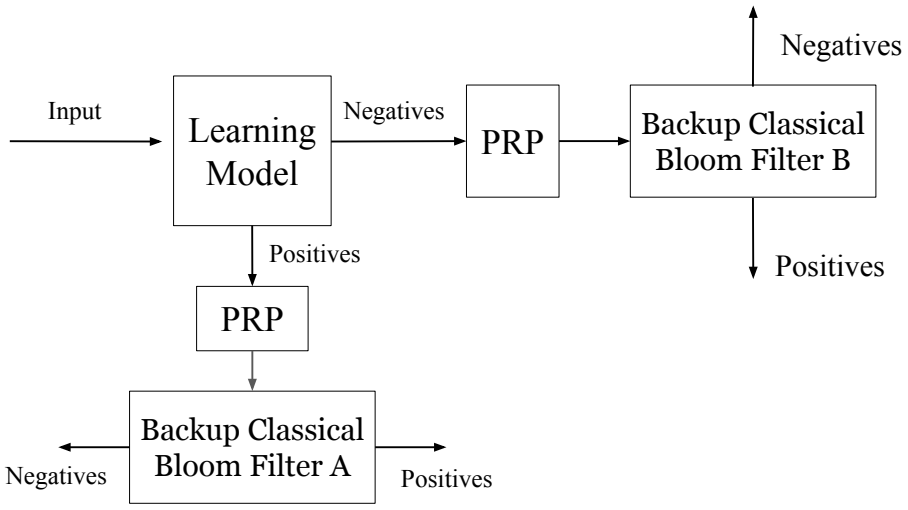


Fig. 4: A Downtown Bodega Filter. Both Positive and Negative results for the Bloom Filter Learning Model are routed to Backup Bloom Filters secured with Pseudo-random Permutations

1. *A Bloom Filter Learning Model* $l_r^{\mathcal{D}}$ *trained on* $\mathcal{D}$
2. *A suitable threshold* τ *for* l
3. $M_r^{S''}$, *the compressed representation of the set* $S'' = \{F_{k_A}(x) : x \in S | l_r^{\mathcal{D}}(x) \geq \tau\}$ *encoded by a Classical Bloom Filter. We refer to this Classical Bloom Filter as Backup Classical Bloom Filter A.*
4. $M_r^{S'}$, *the compressed representation of the set* $S' = \{F_{k_B}(x) : x \in S | l_r^{\mathcal{D}}(x) < \tau\}$ *encoded by a Classical Bloom Filter. We refer to this Classical Bloom Filter as Backup Classical Bloom Filter B.*

The query algorithm $\tilde{Q}_s$ *for the Downtown Bodega Filter is*

$$(l_r^{\mathcal{D}}(x) \geq \tau \wedge Q_s(M_r^{S''}, F_{k_A}(x)) = 1) \vee (l_r^{\mathcal{D}}(x) < \tau \wedge Q_s(M_r^{S'}(F_{k_B}(x)) = 1)$$

Lemma 2. *Let* B_r *be an* (n, ϵ)*-Bloom Filter using* m *bits of memory. If pseudo-random permutations exist, then there exists a negligible function* $\text{negl}(\cdot)$ *such that for security parameter* λ*, Backup Classical Bloom Filter A and Backup Classical Bloom Filter B are* $(n, \epsilon + \text{negl}(\lambda))$*-adversarial reveal resilient Bloom Filters each using* $m^{'} = m + \lambda$ *bits of memory.*

Lemma 2 is just a rephrasing of Theorem 4.8 of Naor and Yogev [8] and follows directly from the theorem. We include a self-contained proof here for completeness. We also make it more evident that the proof holds not just for adversarial resilient Bloom Filters but also for adversarial **reveal** resilient Bloom Filters that provide guarantees under a strictly stronger adversarial model. It is important to note that, while the adversary has access to the internal representation of the Downtown Bodega Filter, the adversary does **not** have access to the secret keys k_A and k_B. This is consistent with the formulation of Naor and Yogev [8] who state at the end of Section 4 in their paper, concerning Theorem 4.8, "Notice that, in all the above constructions only the pseudo-random

function (permutation) key must remain secret. That is, we get the same security even when the adversary gets the entire memory of the Bloom filter except for the PRF (PRP) key."

Proof. Let us consider Backup Classical Bloom Filter A (a proof for Backup Bloom Filter B can be constructed in the same way). Our setup algorithm $\tilde{C}_r$ merely initializes Backup Classical Bloom Filter A with S''. Our query algorithm on input x queries for $x^{'} = F_{k_A}(x)$. The only additional memory required is for storing k_A which is λ bits long.

The completeness follows from the completeness of the Classical Bloom Filter. The resilience of the construction follows from the following argument: consider an experiment where F_{k_A} in Backup Classical Bloom Filter A is replaced by a truly random oracle $\mathcal{R}(\cdot)$. Since x has not been queried, we know that $R(x)$ is a truly random element that was not queried before, and we may think of it as chosen before the initialization of Backup Bloom Filter A. From the soundness of Backup Classical Bloom Filter A, we get that the probability of x being a false positive is at most ϵ.

Now we show that no probabilistic polynomial time (PPT) adversary A can distinguish between the Backup Bloom Filter A we constructed using $\mathcal{R}(\cdot)$ and the Backup Classical Bloom Filter A construction that uses the pseudo-random permutation F_{k_A} by more than a negligible advantage. Suppose that there does exist a non-negligible function $\delta(\lambda)$ such that adversary A can attack Backup Classical Bloom Filter A and find a false positive with probability $\epsilon + \delta(\lambda)$. We can run adversary A on a Backup Classical Bloom Filter A where the oracle is replaced by an oracle that is either random or pseudo-random. We return 1 if A successfully finds a false positive. This implies that we may distinguish between a truly random permutation and a pseudo-random permutation with probability $\geq \delta(\lambda)$. This contradicts the indistinguishability of pseudo-random permutations.

Theorem 7. *Let SB_r be an (n, ϵ)-Sandwiched Learned Bloom Filter using m bits of memory. If pseudo-random permutations exist, then there exists a negligible function* $\text{negl}(\cdot)$ *such that for security parameter λ there exists an $(n, \epsilon + \text{negl}(\lambda))$-adversarial reveal resilient Downtown Bodega Filter, DB_{r,k_A,k_B}, that uses $m^{'} = m + 2\lambda$ bits of memory.*

Proof. We construct a Downtown Bodega Filter DB_{r,k_A,k_B} from a Sandwiched Learned Bloom Filter SB_r as follows. We use the memory budget of the Initial Classical Bloom Filter to construct the Backup Classical Bloom Filter A. We use the memory budget of the Backup Classical Bloom Filter of the Sandwiched Learned Bloom Filter to construct the Backup Classical Bloom Filter B. We do not modify the Bloom Filter Learning Model l which remains trained on dataset $\mathcal{D}$ with threshold τ. We choose keys $k_A, k_B \in \{0,1\}^\lambda$ and use 2λ bits of extra memory to store them.

The completeness of DB_{r,k_A,k_B} follows from 1) the completeness of Backup Classical Bloom Filter B and 2) the fact that any x such that $l(x) < \tau$ is declared to be not in S by $\tilde{Q}_s$ the query algorithm f the Downtown Bodega Filter if and only if the query algorithm of the Classical Bloom Filter with Backup Classical Bloom Filter B, $Q_s(M_r^{S^{'}}, F_{k_B}(x))$, is also 0. This fact follows directly from the query algorithm $\tilde{Q}_s$ for the Downtown Bodega Filter stated in Definition 20.

To prove the resilience of the construction, we first show that the security of the Downtown Bodega Filter construction is reducible to the security of Backup Classical Bloom Filter A and Backup Classical Bloom Filter B. Consider a false positive i.e. an $x \notin S$ for which the Downtown Bodega Filter returns 1. From the definition of the Downtown Bodega Filter, one of the following two cases must be true.

Case 1: The Bloom Filter Learning Model, l, returned a value $\geq \tau$ and Backup Classical Bloom Filter A returned 1, more precisely, $l(x) \geq \tau \wedge Q_s(M_r^{S^{''}}, F_{k_A}(x)) = 1$

Case 2: The Bloom Filter Learning Model, l, returned a value $< \tau$ and Backup Bloom Classical Filter B returned 1, more precisely, $l(x) < \tau \wedge Q_s(M_r^{S^{'}}, F_{k_B}(x)) = 1$

Therefore, for any probabilistic polynomial time (PPT) adversary to induce a false positive in the overall Downtown Bodega Filter construction, they must either induce a false positive in Backup Classical Bloom Filter A or Backup Classical Bloom Filter B. We have already proven in Lemma 2 that both Backup Classical Bloom Filter A and Backup Classical Bloom Filter B are $(n, \epsilon + \text{negl}(\lambda))$-adversarial reveal resilient. It follows that the entire construction is $(n, \epsilon + \text{negl}(\lambda))$-adversarial reveal resilient. This concludes our proof.

4 Discussion

In this section, we discuss the only two known attacks on the Learned Bloom Filter, introduced by Reviriego et al [6]. We refer to Attack 1 from their work as the Blackbox Mutation Attack, and Attack 2 from their work as the Whitebox Mutation Attack respectively. We discuss the Blackbox Mutation Attack in Section 4.1, and the Whitebox Mutation Attack in Section 4.2. Both sections include details on how our Secure Learned Bloom Filter construction mitigates the attack.

4.1 Black-box Mutation Attack

The black-box adversarial model defined by Reviriego et al [6] is slightly weaker but very similar to the adversarial model we define in Challenge 5 of Section 2.2. Both our adversarial model and Reviriego et al's black-box adversary model allow the adversary access to query the Learned Bloom Filter. One major difference is that our model allows the adversary to choose the initial set S that is represented by the Learned Bloom Filter, whereas Reviriego et al's model does not. In their attack, Reviriego et al. first test elements until a positive (whether a false positive or true positive) is found. They then *mutate* the positive by changing a small fraction of the bits in the input to generate more false positives. The attack targets the Bloom Filter Learning Model (recall Figure 2) by making it generate false positives without the input reaching the Backup Bloom Filter. This attack is mitigated by the Downtown Bodega Filter because the Downtown Bodega Filter passes queries through a Secure Classical Backup Bloom Filter even for the case where the Bloom Filter Learning Model returns true i.e when $l(x) > \tau$.

4.2 White-box Mutation Attack

The white-box adversarial model defined by Reviriego et al [6] is similar to the adversarial model we define in Challenge 6 in Section 2.2. With knowledge of the state of the Bloom Filter Learning Model, the adversary can generate mutations in a more sophisticated way. Reviriego et al. provide the example of a malicious URL dataset where an adversary may begin with a non-malicious URL and make changes such as removing the "s" in "https" or removing the "www" to generate false positives. Since we have shown the Downtown Bodega Filter to be (n, t, ϵ)-adversarial **reveal** resilient in the steady setting, such mutations will not provide the adversary any advantage over the construction.

5 Hybrid Adversarial Model

In this section, we first define a hybrid model, where part of the queries are chosen by an adversary, while the rest are non-adversarial ("regular") queries. Next, we analyze the performance of the Downtown Bodega Filter and the Secure Classical Bloom Filter respectively in our hybrid model.

We provide results for the conditions in which the Downtown Bodega Filter outperforms the Secure Classical Bloom Filter in the hybrid model. We then discuss the trade-offs of our approach and provide a realistic example where the Downtown Bodega Filter displays better performance as compared to the Secure Classical Bloom Filter under many settings.

5.1 Hybrid Model

Let $A = (A_C, A_Q)$ be a probabilistic polynomial time (PPT) adversary as defined in Section 2. Consider a set of N queries sent to a (Classical or Learned) Bloom Filter. Our adversary, A, is allowed to choose exactly αN of those queries, where $\alpha \in [0, 1]$. As an example, the N queries may be part of a streaming workload under any of the streaming models described by Muthukrishnan [15].

5.2 Downtown Bodega Filter

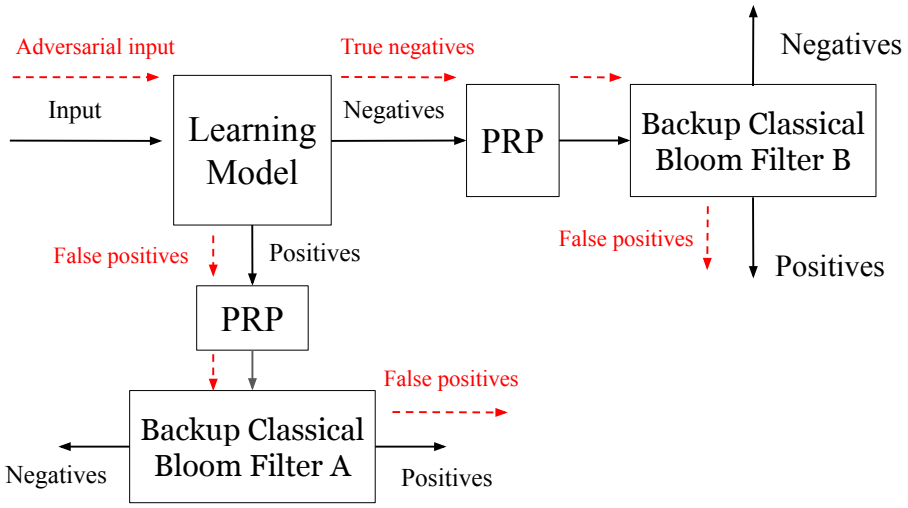


Fig. 5: To generate a false positive in the Downtown Bodega Filter, the adversary must either 1) generate a false positive in the Bloom Filter Learning Model and direct their query through Backup Classical Bloom Filter A or 2) generate a true negative in the Bloom Filter Learning Model and direct their query through Backup Classical Bloom Filter B

Let $DB_r = (\tilde{C}_r, \tilde{Q}_s)$ be a Downtown Bodega Filter. Let l_r be the corresponding Bloom Filter Learning Model and τ its threshold. Let $M_r^{S''}$ be the internal representation of Backup Classical Bloom Filter A and $M_r^{S'}$ be the internal representation of Backup Classical Bloom Filter B. For an adversarial query to generate a false positive in the Downtown Bodega Filter, one of the following must hold for a given query (see Figure 5):

1. The query generates a false positive in the Bloom Filter Learning Model and a false positive in the Backup Bloom Filter A.

2. The query generates a true negative in the Bloom Filter Learning Model and a false positive in the Backup Bloom Filter B

We first model the false positive probability of the Downtown Bodega Filter as a function of its memory budget, m, the set that the Downtown Bodega Filter is encoding, S, and the corresponding training dataset $\mathcal{D}$. Without loss of generality, let α_P of the adversarial queries be such that they generate false positives in the Bloom Filter Learning Model and go through Backup Bloom Filter A. Similarly, let α_N of the adversarial queries be such that they generate true negatives and go through Backup Bloom Filter B. Note that $\alpha = \alpha_P + \alpha_N$.

Let $\text{FPR}(S, m)$ be the expected false positive probability of a Classical Bloom Filter that encodes the set S with memory budget m. Let $\text{FPR}_L(S, \mathcal{D}, m)$ be the expected false positive probability of a Bloom Filter Learning Model L that encodes the set S using the training dataset $\mathcal{D}$ with memory budget m. Similarly, let $\text{TNR}_L(S, \mathcal{D}, m)$ be the expected true negative probability of a Bloom Filter Learning Model L that encodes the set S using the training dataset $\mathcal{D}$ with memory budget m. We assume that the correctness probability of the Bloom Filter Learning Model is independent of the correctness probability of the Backup Classical Bloom Filters. In particular, we assume that for any m, m', $\text{FPR}_L(m) \cap \text{FPR}(m') = \text{FPR}_L(m)\text{FPR}(m')$ and $\text{TNR}_L(m) \cap \text{FPR}(m') = \text{TNR}_L(m)\text{FPR}(m')$.

Consider a system where the total memory budget is M. Let the memory allocation of a Downtown Bodega Filter from memory budget M be assigned as follows. Let m_L be the number of bits of memory assigned to a Bloom Filter Learning Model L. Let m_A be the number of bits of memory assigned to Backup Bloom Filter A. Let m_B be the number of bits of memory assigned to Backup Bloom Filter B. Let λ be the number of bits assigned to the key of the pseudo-random permutations used before Backup Bloom Filter A and Backup Bloom Filter B. Note that to stay within the memory budget it must hold that $M \geq m_L + m_A + m_B + 2\lambda$.

Theorem 8. *For any memory budget M, any set S, and corresponding training dataset $\mathcal{D}$, The Downtown Bodega Filter encoding S provides an expected false positive probability of*

$$FPR_{DB}(S, \mathcal{D}, M) = FPR_L(S, \mathcal{D}, m_L)FPR(S'', m_A) + TNR_L(S, \mathcal{D}, m_L)FPR(S', m_B)$$

Where $S' \subset S$ is the set of elements in S for which the Bloom Filter Learning Model L returns negative, $S'' \subset S$ is the set of elements in S for which the Bloom Filter Learning Model L returns positive, and where the probability is taken over the random coins of the pseudo-random permutations, the random coins used in the construction of the Bloom Filter Learning Model, the random coins used in the construction of Backup Classical Bloom Filters A and B, and the random coins used in the generation of the non-adversarial queries.

Proof. From the definition of a Downtown Bodega Filter (Definition 20, in particular look at the formulation for the query algorithm $\tilde{Q}_s$. Figure 5 is also helpful here) it follows that a false positive in the overall construction must either be a false positive in Backup Classical Bloom Filter A or a false positive in Backup Classical Bloom Filter B. If the query is a false positive in Backup Classical Bloom Filter A, it must also be a false positive in the Bloom Filter Learning Model. Alternatively, if the query is a false positive in Backup Classical Bloom Filter B, it must be a true negative in the Bloom Filter Learning Model. The result follows.

We now derive an expression for the false positive probability of the Downtown Bodega Filter in the hybrid adversarial setting. Recall that we are assuming that out of N queries, the adversary makes α_P queries that generate a false positive in the Bloom Filter Learning Model and α_N queries that generate a true negative in the Bloom Filter Learning Model.

Theorem 9. *In the hybrid adversarial setting, the expected false positive probability of the Downtown Bodega Filter is*

$$\alpha_P FPR(S'', m_A) + \alpha_N FPR(S', m_B) + (1 - \alpha_P - \alpha_N) FPR_{DB}(S, \mathcal{D}, M)$$

Where $S' \subset S$ is the set of elements in S for which the Bloom Filter Learning Model L returns negative, $S'' \subset S$ is the set of elements in S for which the Bloom Filter Learning Model L returns positive, and where the probability is taken over the random coins of the pseudo-random permutations, the random coins used in the construction of the Bloom Filter Learning Model, the random coins used in the construction of Backup Classical Bloom Filters A and B, and the random coins used in the generation of the non-adversarial queries.

Proof. For each query i among N queries, one of the following cases holds.

Case 1: The query is not adversary-generated. Therefore as established by Theorem 8, the False Positive Probability for the query is $\text{FPR}_{DB}(S, \mathcal{D}, M)$. There are $(1 - \alpha_P - \alpha_N)N$ such queries.

Case 2: The query is adversary-generated such that it generates a false positive in the Bloom Filter Learning Model. Since the Bloom Filter Learning Model generating a false positive and the Bloom Filter Learning Model generating a true negative are mutually exclusive events, the False Positive Probability for the query is the False Positive Probability of Backup Classical Bloom Filter A i.e $\text{FPR}(S'', m_A)$. There are $\alpha_P N$ such queries.

Case 3: The query is adversary-generated such that it generates a true negative in the Bloom Filter Learning Model. Following logic similar to case 2, we can derive the False Positive Probability of the query to be $\text{FPR}(S', m_B)$. There are $\alpha_N N$ such queries.

The expected false positive probability of N queries is then $\frac{1}{N}(\alpha_P N \cdot \text{FPR}(S'', m_A) + \alpha_N N \cdot \text{FPR}(S'', m_B) + (1 - \alpha_P - \alpha_N)N \cdot \text{FPR}_{DB}(S, M))$. The statement of the theorem follows.

5.3 Secure Classical Bloom Filter

An alternative construction is to simply use a well-tuned (n, t, ϵ)-adversarial reveal resilient Classical Bloom Filter. We will refer to this construction in this section as the Secure Classical Bloom Filter without confusion. The expected false positive probability of the Secure Classical Bloom Filter encoding a set S is, by definition, $\text{FPR}(S, m_L + m_A + m_B + \lambda)$. The extra λ is because the Secure Classical Bloom Filter requires one less pseudo-random permutation than the Downtown Bodega Filter.

We now provide an expression that encapsulates all the cases in the hybrid adversarial setting where a Downtown Bodega Filter construction provides a lower false positive probability compared to a secure Classical Bloom Filter construction for the same memory budget.

Theorem 10. *For a given memory budget $M = m_L + m_A + m_B + 2\lambda$, any set S and corresponding training dataset $\mathcal{D}$, in the hybrid adversarial setting with given $\alpha = \alpha_P + \alpha_N$, the expected false positive probability of the Downtown Bodega Filter is lower than the expected false positive probability of the Secure Classical Bloom Filter if the following holds:*

$$\begin{aligned} \alpha_P FPR(S'', m_A) + \alpha_N FPR(S', m_B) + (1 - \alpha_P - \alpha_N) FPR_{DB}(S, \mathcal{D}, M) \\ < FPR(S, m_L + m_A + m_B + \lambda) \end{aligned} \tag{2}$$

Where $S' \subset S$ is the set of elements in S for which the Bloom Filter Learning Model L returns negative, $S'' \subset S$ is the set of elements in S for which the Bloom Filter Learning Model L returns

positive, and where the probability is taken over the random coins of the pseudo-random permutations, the random coins used in the construction of the Bloom Filter Learning Model, the random coins used in the construction of Backup Bloom Filters A and B, the random coins used in the construction of the Secure Classical Bloom Filter, and the random coins used in the generation of the non-adversarial queries.

Proof. The proof follows directly from the expression derived for the expected false positive probability of the Downtown Bodega Filter construction in Theorem 9 and the expression for the expected false positive probability of the Secure Classical Bloom Filter.

Parameter	Explanation	Value
M	Total memory budget	2 MB
m_L	Memory budget for Bloom Filter Learning Model	1 MB
m_A	Memory budget for Backup Classical Bloom Filter A	0.5 MB
m_B	Memory budget for Backup Classical Bloom Filter B	0.5 MB
n	Cardinality of set to encode	1.7 Million
c	$\frac{\text{FPR of Bloom Filter Learning Model)}}{\text{FPR of Classical Bloom Filter}}$ for same memory budget	0.25
λ	Number of bits in secret key	128 bits
Q_N	Fraction of true negative non-adversarial queries	0.5

Table 1: A summary of the chosen values for our realistic example of the performance tradeoffs of a Downtown Bodega Filter compared to a Secure Classical Filter in the hybrid adversarial case

5.4 Realistic Example

Mitzenmacher and Broder [4] show that the false positive probability for a Classical Bloom Filter with m bits encoding a set S, using $k(S, m)$ hash functions is

$$\text{FPR}(S, m) = (1 - e^{-k(S,m)\cdot|S|/m})^{k(S,m)}$$

In our analysis, we use the value of the number of hash functions $k(S, m)$ is always optimally chosen to be $k(S, m) = \ln 2 \cdot (m/|S|)$ (this optimal value is also derived by Mitzenmacher and Broder [4]). We further model the false positive rate of a Bloom Filter Learning Model as being the same as the false positive rate of a Classical Bloom Filter encoding set S but with a better false positive probability for the same memory budget. This is consistent with the assumptions made by prior work including Kraska et al [5] and Mitzenmacher [7].

$$\text{FPR}_L(S, \mathcal{D}, m) = c(1 - e^{-k(S,m)\cdot|S|/m})^{k(S,m)}$$

where $c \leq 1$.

We note that the true negative probability of a Bloom Filter Learning Model is merely the probability of a negative entry (which is constant as we are assuming set S is constant) is not marked as a false positive. Let Q_N be the fraction of true negative non-adversarial queries. We have

$$\text{TNR}_L(S, \mathcal{D}, m) = (1 - \text{FPR}_L(S, \mathcal{D}, m))Q_N = (1 - c(1 - e^{-k(S,m)\cdot|S|/m})^{k(S,m)})Q_N$$

To evaluate how much lower the false positive probability of a Bloom Filter Learning Model needs to be for the Downtown Bodega Filter to perform better than the Secure Classical Bloom Filter, we may then use these derivations in Theorem 10.

We choose realistic values for our example from prior work on evaluating Learned Bloom Filters [5] on Google's transparency report. We pick 2 Megabytes as our memory budget, m, chosen from the range of values in Figure 10 of Kraska et al [5]. We choose the cardinality of the set we want to encode, $|S|$, as 1.7 million based on the number of unique URLs in Google's transparency report evaluated in Kraska et al [5]. With a memory budget of 2 Megabytes, Kraska et al [5] demonstrate that a Learned Bloom Filter has 0.25 of the False Positive Ratio of a Classical Bloom Filter, hence we use that as our value for c. We assume Q_N to be 0.5 for this example (results for the complete range of values of Q_N can be found in Appendix B).

We use 128 bits as the size of our security parameter, λ. For the case of the Downtown Bodega Filter, we let the Bloom Filter Learning Model take 1 Megabytes while dividing the remaining 1 Megabytes equally between Backup Classical Bloom Filters A and B. Backup Classical Bloom Filters A and B encode S'' and S' respectively, which are both subsets of the set S (refer to Definition 20).

We take α to be a variable ranging from 0 to 1 equally divided between α_P and α_N (refer to Appendix B for other strategies of partitioning α between α_P and α_N). Our chosen values are summarized in Table 1. Figure 6 shows the results of our calculations. As can be seen, when the adversary has access to less than a certain cutoff fraction of the workload, the Downtown Bodega Filter outperforms the Secure Classical Bloom Filter for the same memory budget. The C source code for our model and analysis can be found in an anonymously hosted code repository [16]. A thorough experimental evaluation of the Hybrid Model across multiple datasets and a comprehensive range of model parameters can be found in Appendix B.

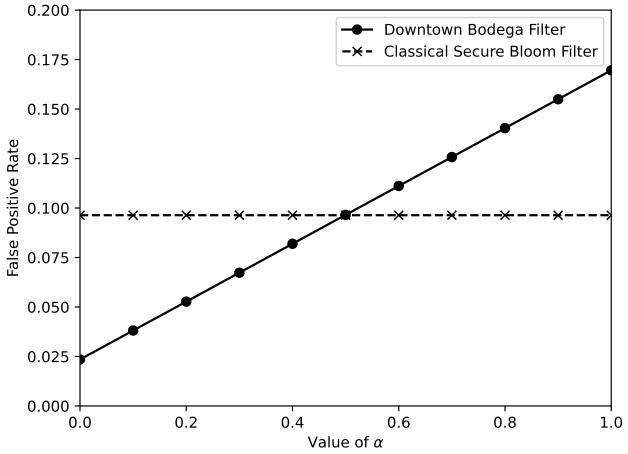


Fig. 6: The FPR of the Downtown Bodega Filter compared to the Secure Classical Bloom Filter in the hybrid adversarial setting as evaluated on the Google transparency report.

6 Open Problems

This work suggests the following three open problems.

Problem 1: Security in the unsteady setting. While we have provided secure constructions for the Learned Bloom Filter, our results only hold for the steady setting, where the query algorithm $\tilde{Q}_s$ may **not** modify the internal representation of the Learned Bloom Filter. Naor and Yogev [8] provide secure constructions for the Classical Bloom Filter in an unsteady setting. We leave the formulation of secure constructions for the Learned Bloom Filter in the unsteady setting as an open problem.

Problem 2: Security in a stronger adversarial model. The adversarial model we consider for the Learned Bloom Filter is stronger than Naor and Yogev [8] in that it allows an adversary access to the internal state of the Bloom Filter, but it is weaker than the *Bet-Or-Pass (BP) test* security notion Naor and Oved [11] as proven in their work. We leave the construction of a Bet-Or-Pass test resilient Learned Bloom Filter as an open problem.

Problem 3: Security when allowing insertions Clayton et al. [12] and Filic et al. [13] prove security results for an adversary that can not only query but also insert entries in the Learned Bloom Filter. We leave the formulation of secure constructions of the Learned Bloom Filter under an adversarial model that allows insertion as an open problem.

Problem 4: Security against computationally unbounded adversaries. In this work, we only consider probabilistic polynomial time (PPT) adversaries. An open problem is to prove or disprove any security guarantees for our constructions against computationally unbounded adversaries. Alternatively, an open problem is to provide secure constructions against computationally unbounded adversaries. Naor and Yogev [8] discuss computationally unbounded adversaries in Section 5 of their work.

Disclosure of Interests. The authors have no competing interests

Bibliography

[1] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, p. 422–426, jul 1970. [Online]. Available: https://doi.org/10.1145/362686.362692

[2] K. Christensen, A. Roginsky, and M. Jimeno, "A new analysis of the false positive rate of a bloom filter," *Information processing letters*, vol. 110, no. 21, pp. 944–949, 2010.

[3] P. Bose, H. Guo, E. Kranakis, A. Maheshwari, P. Morin, J. Morrison, M. Smid, and Y. Tang, "On the false-positive rate of bloom filters," *Information processing letters*, vol. 108, no. 4, pp. 210–213, 2008.

[4] M. Mitzenmacher and A. Broder, "Network applications of bloom filters: A survey," *Internet Mathematics Journal*, 2004.

[5] T. Kraska, A. Beutel, E. H. Chi, J. Dean, and N. Polyzotis, "The case for learned index structures," in *Proceedings of the 2018 International Conference on Management of Data*, ser. SIGMOD '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 489–504. [Online]. Available: https://doi.org/10.1145/3183713.3196909

[6] P. Reviriego, J. Alberto Hernández, Z. Dai, and A. Shrivastava, "Learned bloom filters in adversarial environments: A malicious url detection use-case," in *2021 IEEE 22nd International Conference on High Performance Switching and Routing (HPSR)*, 2021, pp. 1–6.

[7] M. Mitzenmacher, "A model for learned bloom filters, and optimizing by sandwiching," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, ser. NIPS'18. Red Hook, NY, USA: Curran Associates Inc., 2018, p. 462–471.

[8] M. Naor and Y. Eylon, "Bloom filters in adversarial environments," *ACM Trans. Algorithms*, vol. 15, no. 3, jun 2019. [Online]. Available: https://doi.org/10.1145/3306193

[9] S. Tarkoma, C. E. Rothenberg, and E. Lagerspetz, "Theory and practice of bloom filters for distributed systems," *IEEE Communications Surveys & Tutorials*, vol. 14, no. 1, pp. 131–155, 2012.

[10] T. Gerbet, A. Kumar, and C. Lauradoux, "The power of evil choices in bloom filters," in *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2015, pp. 101–112.

[11] M. Naor and N. Oved, "Bet-or-pass: Adversarially robust bloom filters," in *Theory of Cryptography*, E. Kiltz and V. Vaikuntanathan, Eds. Cham: Springer Nature Switzerland, 2022, pp. 777–808.

[12] D. Clayton, C. Patton, and T. Shrimpton, "Probabilistic data structures in adversarial environments," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 1317–1334. [Online]. Available: https://doi.org/10.1145/3319535.3354235

[13] M. Filic, K. G. Paterson, A. Unnikrishnan, and F. Virdia, "Adversarial correctness and privacy for probabilistic data structures," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 1037–1050. [Online]. Available: https://doi.org/10.1145/3548606.3560621

[14] J. Katz and Y. Lindell, *Introduction to Modern Cryptography, Second Edition*, 2nd ed. Chapman & Hall/CRC, 2014.

[15] S. Muthukrishnan, "Data streams: algorithms and applications," *Found. Trends Theor. Comput. Sci.*, vol. 1, no. 2, p. 117–236, aug 2005. [Online]. Available: https://doi.org/10.1561/0400000002

[16] "Open Source Downtown Bodega Filter Implementation," https://codeberg.org/h_research/adversary-resilient-learned-bloom-filters, 2024.
[17] A. Sato and Y. Matsui, "Fast partitioned learned bloom filter," in *Proceedings of the 37th International Conference on Neural Information Processing Systems*, ser. NeurIPS '23. Red Hook, NY, USA: Curran Associates Inc., 2024.
[18] K. Vaidya, E. Knorr, M. Mitzenmacher, and T. Kraska, "Partitioned learned bloom filters," in *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. [Online]. Available: https://openreview.net/forum?id=6BRLOfrMhW
[19] Z. Dai and A. Shrivastava, "Adaptive learned bloom filter (ada-bf): Efficient utilization of the classifier with application to real-time information filtering on the web," in *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., 2020. [Online]. Available: https://proceedings.neurips.cc/paper/2020/hash/86b94dae7c6517ec1ac767fd2c136580-Abstract.html
[20] "Malicious URLs Dataset," https://www.kaggle.com/datasets/sid321axn/malicious-urls-dataset.
[21] H. S. Anderson and P. Roth, "Ember: An open dataset for training static pe malware machine learning models," 2018. [Online]. Available: https://arxiv.org/abs/1804.04637

Appendices

Appendix A contains a brief but self-contained treatment of pseudo-random permutations. Appendix B contains detailed results across a wide range of model parameters for the Hybrid Model.

A Pseudo-random Permutations

This section provides a brief self-contained treatment of pseudo-random permutations adapted from Chapter 3 of Katz Lindell [14] here.

Let Perm_n be the set of all permutations on $\{0,1\}^n$.

Definition 21. *Let an efficient permutation F be any permutation for which there exists a polynomial time algorithm to compute $F_k(x)$ given k and x, and there also exists a polynomial time algorithm to compute $F_k^{-1}(x)$ given k and x.*

Definition 22. *Let $F : \{0,1\}^* \times \{0,1\}^* \mapsto \{0,1\}^*$ be an efficient, length-preserving, keyed function. F is a keyed permutation if $\forall k$, $F_k(\cdot)$ is one-to-one.*

Definition 23. *Let $F : \{0,1\}^* \times \{0,1\}^* \mapsto \{0,1\}^*$ be an efficient keyed permutation. F is a pseudo-random permutation if for all probabilistic polynomial time distinguishers D, there exists a negligible function* `negl`*, such that*

$$|\Pr[D^{F_k(\cdot)F_k^{-1}(\cdot)}(1^n) = 1] - \Pr[D^{f_n(\cdot)f_n^{-1}(\cdot)}(1^n) = 1]| \leq \texttt{negl}(n)$$

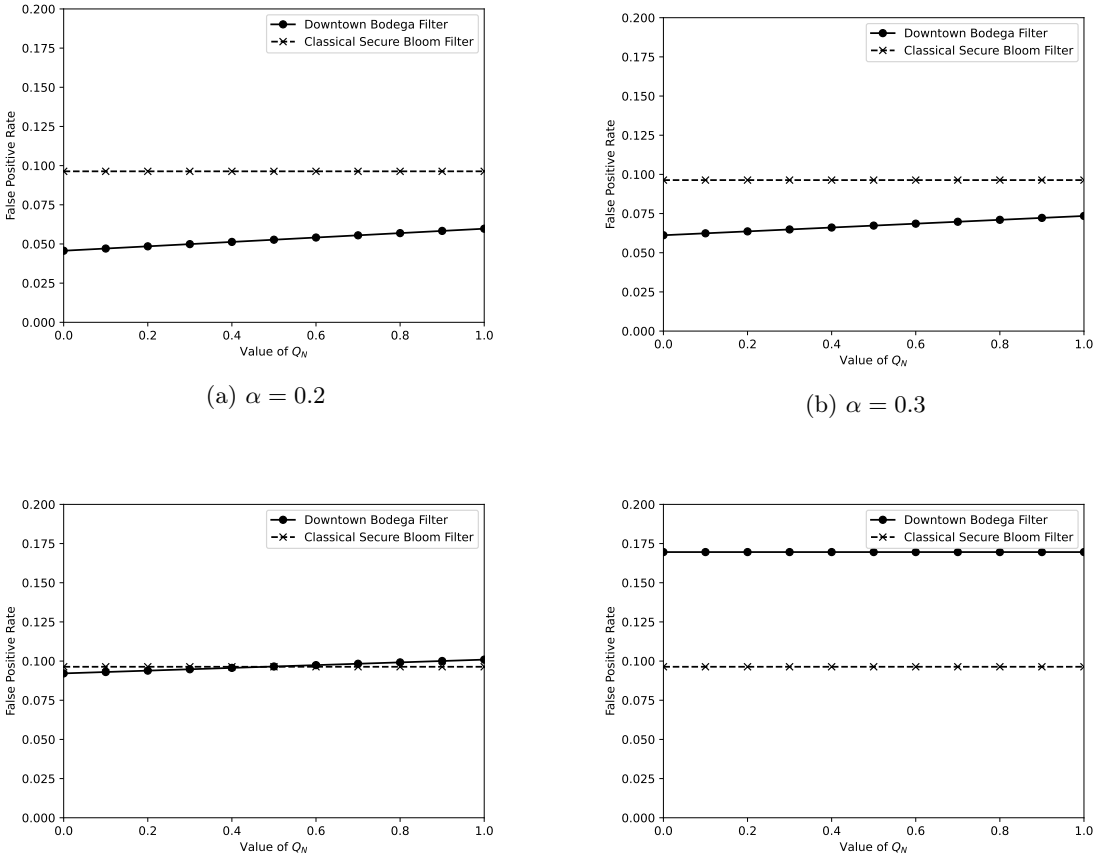
where the first probability is taken over uniform choice of $k \in \{0,1\}^n$ and the randomness of D, and the second probability is taken over uniform choice of $f \in Perm_n$ and the randomness of D.

B Hybrid Model - Additional Experiments

This section contains additional experiments under the Hybrid Adversarial Model setting described in Section 5. We analyze the Hybrid Model using 494 lines of `C` which we have open-sourced [16].

B.1 Varying Non-Adversarial True Negatives (Q_N)

In the main text, we assumed the fraction of non-adversarial queries that were true negatives, Q_N to be 0.5. Here we show the results for the entire range of values of $Q_N \in [0, 1]$ for α taking 4 values: $0.2, 0.3, 0.5, 1.0$, with each value partitioned equally between α_P and α_N. The results are in Figure 7.



(a) $\alpha = 0.2$

(b) $\alpha = 0.3$

(c) $\alpha = 0.5$

(d) $\alpha = 1.0$

Fig. 7: The FPR of the Downtown Bodega Filter compared to the Secure Classical Bloom Filter in the hybrid adversarial setting as evaluated on the Google transparency report with Q_N taking a range of values in the interval $[0, 1]$

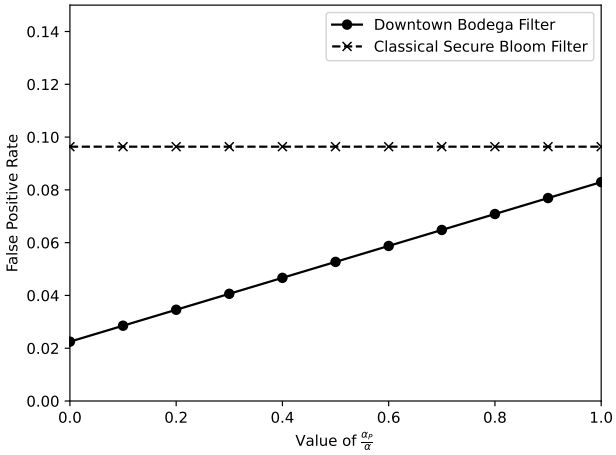


Fig. 8: The FPR of the Downtown Bodega Filter compared to the Secure Classical Bloom Filter in the hybrid adversarial setting as evaluated on the Google transparency report as we vary the fraction of adversarial queries that generate false positives

B.2 Adversary Strategies

In the main text, we explored the case where the adversary divides αN queries equally between queries that generate False Positives and queries that generate False Negatives. We conduct an experiment here for all partitions of α between α_P and α_N for $\alpha = 0.2$ to see how the False Positive Rate of the Downtown Bodega Filter is impacted. We vary the fraction of α assigned to α_P from 0 to 1. Here, 0 means the adversary spends their entire budget of αN queries on true negatives, and 1 means the adversary spends their entire budget of αN queries on false positives. Our earlier experiments are setting this fraction to 0.5, in this framework. The results are in Figure 8

B.3 Additional Datasets

We conduct experiments on two other common evaluation datasets used in prior work on the Learned Bloom Filter [17, 18, 19]. We include a brief description of these datasets, adapted from Sato and Matsui's recent NeurIPS 2023 work on Fast Partitioned Learned Bloom Filters [17].

- **Malicious URLs Dataset** [20]: The URLs dataset contains $223,088$ malicious and $428,118$ benign URLs.
- **EMBER Dataset** [21]: This dataset contains $300,000$ malicious and $400,000$ benign files.

For these datasets, we change the values of the cardinality of the set to encode, n, in Table 1. For all other model parameters, we use the same ones listed in the table. Figures 9 and 10 show the results for the Malicious URLs Dataset and the EMBER dataset respectively.

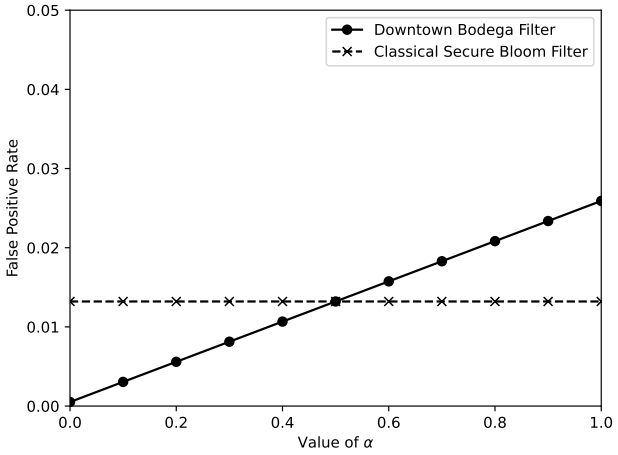


Fig. 9: The FPR of the Downtown Bodega Filter compared to the Secure Classical Bloom Filter in the hybrid adversarial setting as evaluated on the Malicious URLs dataset.

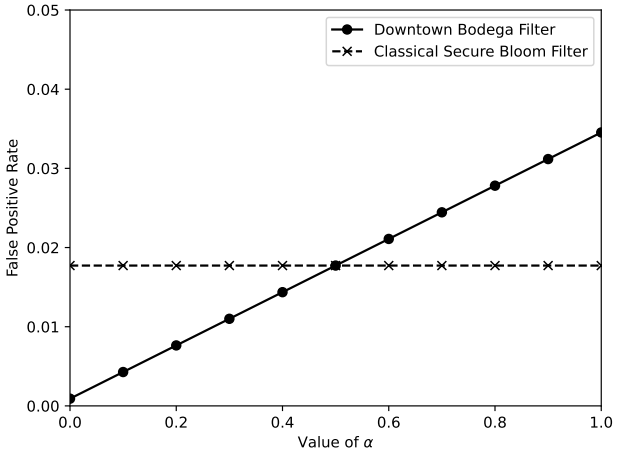


Fig. 10: The FPR of the Downtown Bodega Filter compared to the Secure Classical Bloom Filter in the hybrid adversarial setting as evaluated on the EMBER dataset.