



Adversary Resilient Learned Bloom Filters

Ghada Almashaqbeh¹(✉), Allison Bishop^{2,3}, and Hayder Tirmazi³(✉)

¹ University of Connecticut, Storrs, USA
ghada@uconn.edu

² Proof Trading, New York, USA
abishop@ccny.cuny.edu

³ City College of New York, New York, USA
hayder@alumni.harvard.edu

Abstract. A learned Bloom filter (LBF) combines a classical Bloom filter (CBF) with a learning model to reduce the amount of memory needed to represent a given set while achieving a target false positive rate (FPR). Provable security against adaptive adversaries that advertently attempt to increase FPR has been studied for CBFs, but not for LBFs. In this paper, we close this gap and show how to achieve adaptive security for LBFs. In particular, we define several adaptive security notions capturing varying degrees of adversarial control, including full and partial adaptivity, in addition to LBF extensions of existing adversarial models for CBFs, including the Always-Bet and Bet-or-Pass notions. We propose two secure LBF constructions, PRP-LBF and Cuckoo-LBF, and formally prove their security under these models assuming the existence of one-way functions. Based on our analysis and use case evaluations, our constructions achieve strong security guarantees while maintaining competitive FPR and memory overhead.

1 Introduction

Bloom filters are probabilistic data structures that allow building a succinct representation of a data set while offering approximate membership queries, i.e., whether an element x is in a set S . Bloom filters have one-sided error guarantees: if $x \in S$ the filter always returns yes (i.e., no false negatives), but if $x \notin S$, the filter may return yes, instead of no, with some probability resulting in a false positive. Bloom filters greatly improve space efficiency; instead of storing the full set, this set is encoded in a much shorter bit string using a family of hash functions to map each item into a few bits (set to 1) in this string [3].

Learned Bloom filters (LBFs) have been introduced to improve upon conventional, or classical Bloom filters (CBFs) [14]. In particular, and as shown in Fig. 1, an LBF combines a learning model with a CBF to obtain a lower false positive rate (FPR) than CBFs under the same memory budget. The learning model acts as an initial filter, providing a probabilistic estimate on whether $x \in S$, while a smaller CBF serves as a backup to prevent any false negatives.

Bloom filters are used in various practical applications, including Linux network drivers, network protocols, deep packet scanners, peer-to-peer networking,

and caching [3, 28]. Many well-known systems, such as Apache Hadoop, Apache HBase, Google BigTable, Google LevelDB, and Meta RocksDB, use Bloom filters as part of their implementations. As LBFs provide the same one-sided error guarantees as CBFs, the applications of LBFs and CBFs are identical. Recently, Roblox used LBFs to achieve 25% cost savings in their production workloads for Spark join queries [16].

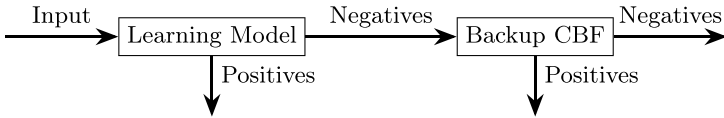


Fig. 1. Conventional LBF architecture—a backup CBF only checks values that are identified as (highly probably) negative by the learning model.

In such critical infrastructures, adversaries may attempt to craft false positives, causing the Bloom filter to deviate from its expected behavior [10], thereby disrupting system operation. For example, as discussed in [20], in a Bloom filter-based spam email whitelist (that stores addresses of known senders), crafting false positives allows spam emails to pass as benign emails. Similarly, in web caches, a Bloom filter can be used to represent the web pages in the cache; a false positive causes an unsuccessful cache access, which degrades performance and may eventually cause a denial of service attack. Another case is related to databases [29], including Meta’s RocksDB and Google’s LevelDB; crafting false positives in the Bloom filters of RocksDB (respectively LevelDB) can degrade the performance of query lookups by up to 8x (respectively 2x).

These cases attest to the importance of studying provable security for Bloom filters against adaptive adversaries. This has been done for CBFs, including studying practical attacks [10], and formalizing notions for adaptive security alongside showing new CBF constructions that realize these notions [20, 21]. However, to the best of our knowledge, no such notions/provably-secure constructions exist for LBFs. Indeed, as an LBF contains a backup CBF, it inherits all the adversarial vulnerabilities of CBFs. However, securing the backup CBF does not imply that the LBF employing this CBF is secure, as LBFs have different designs. That is, having a learning model allows adversaries to craft false positives in new ways [23]. Examples include mutating an existing false positive or modifying certain features (relevant to the learning model) of a true negative input to convert it into a false positive. Nonetheless, the work [23] only demonstrated such attacks while leaving provable adaptive security of LBFs as an open problem.

1.1 Contributions

In this paper, we close this gap and initiate a formal study on the security of LBFs against adaptive adversaries. In particular, we make the following contributions.

New Notions for Adaptive Security. We define several new security notions for LBFs that capture a spectrum of adversarial capabilities. These include *fully-adaptive* adversaries who choose all queries, *partially-adaptive* adversaries who choose a fraction of queries, *bet-or-pass* adversaries who can selectively choose to output a guess in the security game, and *always-bet* adversaries who always output a guess (the latter two are inspired by similar models defined for CBFs [21]). These models provide fine-grained understanding of the attack surface against LBFs and several security-efficiency tradeoffs. Moreover, we explore the relationships between these various notions, proving which security notions imply which. We believe that our notions contribute to establishing a foundation for analyzing learning-augmented data structures in adversarial settings.

Constructions. We propose two constructions for LBFs: PRP-LBF and Cuckoo-LBF. Our constructions extend a prior LBF variant called Partitioned LBFs [30] by introducing cryptographic hardness through pseudorandom permutations (PRPs) and pseudorandom functions (PRFs). Assuming one-way functions exist, we show that PRP-LBF achieves security under the fully-adaptive and partially-adaptive adversarial models. We also show (assuming one-way functions) that Cuckoo-LBF satisfies the bet-or-pass security notion (the extended notion for LBFs). Bet-or-pass is one of the strongest adversaries considered by prior work [15, 21] for CBFs. Our constructions require only $\mathcal{O}(n \log \frac{1}{\varepsilon} + \lambda)$ additional bits of memory, where n is the cardinality of the set represented by the filter, ε is the desired FPR, and λ is the security parameter.

Performance Evaluation. We analyze FPR of our constructions in both adversarial and non-adversarial settings. Our results show that LBFs can provide lower FPR than CBFs in realistic workloads using the same memory budget. They also show that our LBF constructions achieve better FPR-memory tradeoffs than prior CBF constructions while maintaining strong security guarantees. These results demonstrate that adversarial resilience can be supported for LBFs with minimal performance overhead.

1.2 Related Work

Security of Classical Bloom Filters. Gerbet et al. [10] demonstrate practical attacks on CBFs in the context of web crawlers and spam email filtering, and propose combining universal hash functions with message authentication codes to mitigate a subset of these attacks. Naor et al. [20] define an adversarial model for CBFs and show that for computationally bounded adversaries, non-trivial adversary resilient CBFs exist if and only if one-way functions exist, and that for computationally unbounded adversaries there exists a CBF that is secure when the adversary makes t queries while using only $\mathcal{O}(n \log \frac{1}{\varepsilon} + t)$ bits of memory.

Clayton et al. [4] and Filic et al. [8] present stronger adversarial models than [20], giving the adversary the capability of performing insertions and accessing the internal CBF state, and they show secure CBF constructions realizing these models. Naor and Oved [21] introduce a comprehensive study of CBF security and define several robustness notions in a generalized adversarial model.

Concretely, [21] calls the notion of [20] as *Always-Bet (AB)* test since the adversary must output a guess. Then, it introduces a new, strictly stronger, security notion called the *Bet-Or-Pass (BP)* test, giving the adversary the option to pass without outputting a guess. Lotan and Naor [15] provide further results related to relationships between the security notions of [21].

Security of Learned Bloom Filters. LBFs were first introduced by Kraska et al. [14] who showed that LBFs can offer better FPR vs. memory tradeoffs than CBFs. Mitzenmacher [18] provides the first rigorous mathematical model for LBFs, focusing on analyzing performance in terms of memory size and FPR, and introduces an LBF variant called Sandwiched LBF. Vaidia et al. [30] introduce another LBF variant, called partitioned LBFs (PLBFs). PLBFs use a learning model to partition the set S into $p \in \mathbb{N}$ partitions and use a separate backup CBF for each of the p partitions. PLBFs reduce the FPR for specific partitions compared to original LBFs that employ only one backup CBF.

All these works only examine performance, but not security. We are only aware of one prior work that studied LBFs in adversarial settings: Reviriego et al. [23] demonstrate practical attacks on LBFs, which, as mentioned earlier, rely on exploiting the learning model to craft false positives in new ways. In response, they propose two potential mitigations: the first relies on switching back to a CBF once an attack is detected, while the second adds a second backup CBF. However, Reviriego et al. [23] do not provide provable security guarantees for the suggested mitigations, or even empirical evaluations of their effectiveness, and leave provable security of LBFs as an open problem.

1.3 Future Directions

Our work offers important foundational steps towards understanding provable security of LBFs. Here, we list some natural directions for future work.

Unsteady Setting. We provide secure constructions for LBFs in which the query algorithm *does not* modify the internal representation of the LBF. Naor and Yogev [20] call this the *steady* setting, and show secure CBF constructions for the *unsteady* setting where the query algorithm changes the internal filter representation after each query. An interesting direction is to build LBF constructions and prove their adaptive security in the unsteady setting.

Dynamic Bloom Filters. As opposed to static Bloom filters, which do not modify the input set S after construction, dynamic Bloom filters allow insertions of new elements $S' = \{x\} \cup S$ after construction. Note that to maintain their one-sided error guarantees, even dynamic Bloom filters do not allow deletions. Similar to prior work [15, 20, 21] that consider static CBFs, our work focuses on static LBFs. Clayton et al. [4] and Filic et al. [8] show provably-secure security constructions for dynamic CBFs. As such, another interesting direction to explore is formulating adaptive security and building secure constructions for dynamic LBFs under an adversarial model that admits insertions.

Computationally Unbounded Adversaries. Our constructions are secure against polynomial-time adversaries. Another interesting avenue for future research is showing LBF constructions that realize our security notions while assuming computationally unbounded adversaries.

2 Preliminaries

In this section, we review CBFs and their adversarial models from [20,21], which form the basis for the adversarial models we define for LBFs.

Notation. For a set S , $|S|$ denotes the size of S , and $x \leftarrow_s S$ denotes that x is sampled uniformly at random from S . For $n \in \mathbb{N}$, $[n]$ denotes $\{1, \dots, n\}$. Bloom filters can store elements from a finite domain \mathfrak{D} , so we have $S \subseteq \mathfrak{D}$. Lastly, λ denotes the security parameter, $\text{negl}(\lambda)$ denotes a function negligible in λ , and PPT denotes probabilistic polynomial time.

CBF Modeling. We adopt the CBF model from [20] while considering the *steady* setting in which the query algorithm does not change the filter representation.

Definition 1. A Bloom filter $\mathbf{B} = (\mathbf{B}_1, \mathbf{B}_2)$ is a pair of polynomial time algorithms. \mathbf{B}_1 is a randomized construction algorithm that takes as input a set $S \subseteq \mathfrak{D}$ of size n , and outputs a representation M . \mathbf{B}_2 is a deterministic query algorithm that takes as input an element $x \in \mathfrak{D}$ and a representation M , and outputs 1 indicating that $x \in S$, and 0 otherwise. We say that \mathbf{B} is an (n, ε) -Bloom filter if for all sets $S \subseteq \mathfrak{D}$ of cardinality n , the following hold:

1. Completeness: $\forall x \in S : \Pr[\mathbf{B}_2(\mathbf{B}_1(S), x) = 1] = 1$
2. Soundness: $\forall x \notin S : \Pr[\mathbf{B}_2(\mathbf{B}_1(S), x) = 1] \leq \varepsilon$

where the probabilities are taken over the random coins of \mathbf{B}_1 .

A standard CBF is implemented as a string str of length $n_b \in \mathbb{N}$ bits indexed over $[n_b]$, along with $n_h \in \mathbb{N}$ hash functions h_i , for $i \in [n_h]$, from a universal hash family \mathcal{H} used for element mapping. That is, each h_i maps an element from \mathfrak{D} to an index value within $[n_b]$, so $h_i : \mathfrak{D} \mapsto [n_b]$. Then, for each element $x \in S$, for $i \in [n_h]$, the bit at index $h_i(x)$ of str is set to 1 (if it is already set to 1, it stays 1). For a queried element x , a CBF returns 1 if all bits within str at indices corresponding to $h_i(x)$ are set to 1, otherwise, the CBF returns 0.

Always-Bet Security. Naor et al. [20,21] introduced the Always-Bet security game for CBFs, denoted as ABGame , in which an adversary who has oracle access to the CBF aims to find a false positive. It is denoted as always-bet since the adversary is required to output a guess x^* at the end of the game that will be tested whether it is a false positive (i.e., always bets that x^* is a false positive). In detail, for an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, \mathcal{A}_1 chooses a set $S \subseteq \mathfrak{D}$ for which \mathbf{B}_1 will compute a representation M , and \mathcal{A}_2 gets S as input and attempts to find a false positive x^* that was not queried before, given only oracle access $\mathbf{B}_2(M, \cdot)$.

$\text{ABGame}(\mathcal{A}, t, \lambda)$:

1. $S \leftarrow \mathcal{A}_1(1^\lambda)$ such that $S \subseteq \mathfrak{D}$ and $|S| = n$.
2. $M \leftarrow \mathbf{B}_1(S)$.
3. $x^* \leftarrow \mathcal{A}_2^{\mathbf{B}_2(M, \cdot)}(1^\lambda, S)$ where \mathcal{A}_2 can make at most t queries to $\mathbf{B}_2(M, \cdot)$ —in this and other games, we denote the adversarial queries as $\{x_1, \dots, x_t\}$.
4. If $x^* \notin S \cup \{x_1, \dots, x_t\}$ and $\mathbf{B}_2(M, x^*) = 1$, return 1. Otherwise, return 0.

We use the following security notion for all security games discussed in this paper (except for BPGame , which we discuss next).

Definition 2. A Bloom filter \mathbf{B} is (n, t, ε) -secure under a security game Game if for all large enough $\lambda \in \mathbb{N}$, all PPT adversaries \mathcal{A} , and all sets $S \subseteq \mathfrak{D}$ of cardinality n , it holds that $\Pr[\text{Game}(\mathcal{A}, t, \lambda) = 1] \leq \varepsilon$, where the probability is taken over the random coins of \mathbf{B} and \mathcal{A} .

Bet-or-Pass Security. Naor and Oved [21] introduce a security game stronger than ABGame called Bet-or-Pass, or BPGame . Here, \mathcal{A} can either output x^* or *pass*, so it is not always betting on the output of the game to be a false positive. BPGame also defines \mathcal{A} 's *profit* $C_{\mathcal{A}}$, rewarding \mathcal{A} if x^* is a false positive, penalizing \mathcal{A} if x^* is not a false positive, and leaving $C_{\mathcal{A}}$ unchanged if \mathcal{A} chooses to pass.

$\text{BPGame}(\mathcal{A}, t, \lambda)$:

1. $S \leftarrow \mathcal{A}_1(1^\lambda)$ such that $S \subseteq \mathfrak{D}$ and $|S| = n$.
2. $M \leftarrow \mathbf{B}_1(S)$.
3. $(b, x^*) \leftarrow \mathcal{A}_2^{\mathbf{B}_2(M, \cdot)}(1^\lambda, S)$ where \mathcal{A}_2 can make at most t queries, $\{x_1, \dots, x_t\}$, to $\mathbf{B}_2(M, \cdot)$.
4. Return \mathcal{A} 's profit, $C_{\mathcal{A}}$, which is defined as

$$C_{\mathcal{A}} = \begin{cases} 0, & \text{if } b = 0 \\ \varepsilon^{-1}, & \text{if } b = 1 \text{ and } x^* \notin S \cup \{x_1, \dots, x_t\} \\ & \text{and } \mathbf{B}_2(M, x^*) = 1 \\ -(1 - \varepsilon)^{-1}, & \text{otherwise} \end{cases}$$

The profit formulation is set in this way to ensure that an adversary that makes a random guess (which will be a false positive with probability ε) has an expected profit of zero [21]. \mathcal{A} breaks the security of the CBF if its profit is noticeably larger than zero. Thus, the security guarantee for BPGame is defined as an upper bound on the expectation of the adversary's profit.

Definition 3. A Bloom filter \mathbf{B} is (n, t, ε) -secure under BPGame if for all large enough $\lambda \in \mathbb{N}$, all PPT adversaries \mathcal{A} , and all sets $S \subseteq \mathfrak{D}$ of cardinality n , there exists a negligible function $\text{negl}(\cdot)$ such that $\mathbb{E}[C_{\mathcal{A}}] \leq \text{negl}(\lambda)$, where the expectation is taken over the random coins of \mathbf{B} and \mathcal{A} .

3 Definitions and Adversarial Models for LBFs

In this section, we first define a model for LBFs, followed by three LBF adaptive security notions: full adaptivity (which when slightly modified captures always-bet security), partial adaptivity, and learned bet-or-pass.

3.1 LBF Definition

We present our definitions for LBFs, which are based on the model of [18] with additional formalism and adaptations to make it convenient to compare with the models of CBFs introduced earlier. An LBF uses a learning model trained over the dataset the LBF represents, such that the model determines a function \mathcal{L} that models this set. In particular, on input x , \mathcal{L} outputs the probability that $x \in S$. In what follows, we define the notion of a training dataset and a learning model in the context of LBFs (where y_i is a label stating whether $x_i \in S$).

Definition 4. Let $S \subseteq \mathfrak{D}$ be any set represented by a Bloom filter. For any two sets $P \subseteq S$ and $N \subseteq \mathfrak{D} \setminus S$, the training dataset is the set $\mathcal{T} = \{(x_i, y_i = 1) \mid x_i \in P\} \cup \{(x_i, y_i = 0) \mid x_i \in N\}$.

Definition 5. For an $\mathcal{L} : \mathfrak{D} \mapsto [0, 1]$ and threshold τ , we say \mathcal{L} is an $(S, \tau, \varepsilon_p, \varepsilon_n)$ -learning model if for any set $S \subseteq \mathfrak{D}$ the following hold:

1. P-Soundness: $\forall x \notin S : \Pr[\mathcal{L}(x) \geq \tau] \leq \varepsilon_p$
2. N-Soundness: $\forall x \in S : \Pr[\mathcal{L}(x) < \tau] \leq \varepsilon_n$

where the probability is taken over the random coins of \mathcal{L} .

Now, we define an LBF capturing both the classic and learning components. We consider the *steady* setting in which the query algorithm \mathbf{B}_2 does not change the learned representation of the set S (including both the set representation held by the backup CBF, i.e., M , and (\mathcal{L}, τ)).

Definition 6. An LBF $\mathbf{B} = (\mathbf{B}_1, \mathbf{B}_2, \mathbf{B}_3, \mathbf{B}_4)$ is a tuple of four polynomial-time algorithms: \mathbf{B}_1 is as before, \mathbf{B}_2 is a query algorithm, \mathbf{B}_3 is a randomized algorithm that takes a set $S \subseteq \mathfrak{D}$ as input and outputs a training dataset \mathcal{T} , and \mathbf{B}_4 is a randomized algorithm that takes the training dataset \mathcal{T} as input and returns a learning model \mathcal{L} and a threshold $\tau \in [0, 1]$. The internal representation of an LBF contains two components: the classical component M and the learned component (\mathcal{L}, τ) . \mathbf{B}_2 takes as inputs an element $x \in \mathfrak{D}$, M , and (\mathcal{L}, τ) , and outputs 1, indicating that $x \in S$, and 0 otherwise. We say that \mathbf{B} is an $(n, \tau, \varepsilon, \varepsilon_p, \varepsilon_n)$ -LBF if for all sets $S \subseteq \mathfrak{D}$ of cardinality n , it holds that:

1. Completeness: $\forall x \in S : \Pr[\mathbf{B}_2(\mathbf{B}_1(S), \mathbf{B}_4(\mathbf{B}_3(S)), x) = 1] = 1$.
2. Filter soundness: $\forall x \notin S : \Pr[\mathbf{B}_2(\mathbf{B}_1(S), \mathbf{B}_4(\mathbf{B}_3(S)), x) = 1] \leq \varepsilon$.
3. Learning model soundness: $\mathbf{B}_4(\mathbf{B}_3(S))$ is an $(S, \tau, \varepsilon_p, \varepsilon_n)$ -learning model.

where the probabilities are over the random coins of \mathbf{B}_1 , \mathbf{B}_3 , and \mathbf{B}_4 .

Standard LBFs (Fig. 1) use a learning model as a pre-filter before a CBF. The CBF is called a backup CBF as it is only queried on inputs x for which the learning model decides that x is not an element of the stored set S , i.e., $\mathcal{L}(x) < \tau$.

3.2 Full Adaptive Security

Full adaptivity means that \mathcal{A} chooses all the queries submitted to the query algorithm \mathbf{B}_2 , i.e., the adversary controls the entire workload. For CBFs, the full adaptivity game, besides the oracle access to $\mathbf{B}_2(M, \cdot)$, gives oracle access to \mathbf{B}_1 enabling \mathcal{A} to obtain M for any set S of its choosing. Based on that, we define the following game capturing full adaptive security for CBFs.

$\text{FAGame}(\mathcal{A}, t, \lambda)$:

1. $S \leftarrow \mathcal{A}_1(1^\lambda)$ such that $S \subseteq \mathcal{D}$ and $|S| = n$.
2. $M \leftarrow \mathbf{B}_1(S)$.
3. $x^* \leftarrow \mathcal{A}_2^{\mathcal{O}}(1^\lambda, S)$, where $\mathcal{O} = \{\mathbf{B}_1(\cdot), \mathbf{B}_2(M, \cdot)\}$. \mathcal{A}_2 can make at most t queries to \mathbf{B}_2 , and any polynomial number of queries to \mathbf{B}_1 .
4. If $x^* \notin S \cup \{x_1, \dots, x_t\}$ and $\mathbf{B}_2(M, x^*) = 1$, return 1, else, return 0.

For LBFs, we define a similar security game, denoted LFAGame . The difference is that \mathcal{A} now has oracle access to the additional algorithms in the LBF structure. We note that our results hold even if we let \mathcal{A} choose \mathcal{T} directly (rather than having the challenger invoke \mathbf{B}_3 over the set S chosen by \mathcal{A}), as long as the challenger validates that this \mathcal{T} satisfies Definition 4.

$\text{LFAGame}(\mathcal{A}, t, \lambda)$:

1. $S \leftarrow \mathcal{A}_1(1^\lambda)$ such that $S \subseteq \mathcal{D}$ and $|S| = n$.
2. $M \leftarrow \mathbf{B}_1(S)$, $\mathcal{T} \leftarrow \mathbf{B}_3(S)$, and $(\mathcal{L}, \tau) \leftarrow \mathbf{B}_4(\mathcal{T})$.
3. $x^* \leftarrow \mathcal{A}_2^{\mathcal{O}}(1^\lambda, S)$, where $\mathcal{O} = \{\mathbf{B}_1(\cdot), \mathbf{B}_2(M, \mathcal{L}, \tau, \cdot), \mathbf{B}_3(\cdot), \mathbf{B}_4(\cdot)\}$. \mathcal{A}_2 can make at most t queries to \mathbf{B}_2 , and any polynomial number of queries to each of \mathbf{B}_1 , \mathbf{B}_3 , and \mathbf{B}_4 .
4. If $x^* \notin S \cup \{x_1, \dots, x_t\}$ and $\mathbf{B}_2(M, \mathcal{L}, \tau, x^*) = 1$, return 1, else, return 0.

If we remove \mathcal{A}_2 's oracle access to $\mathbf{B}_1(\cdot)$, $\mathbf{B}_3(\cdot)$, and $\mathbf{B}_4(\cdot)$, we obtain a notion for the *always-bet* security game, which we refer to as $\text{LABGame}(\mathcal{A}, t, \lambda)$.

3.3 Partial Adaptive Security

For partial adaptivity, among the t queries to \mathbf{B}_2 , \mathcal{A} can choose αt of them, where $\alpha \in [0, 1]$. These t queries may be part of a batch workload or a streaming workload under any streaming models described by [19]. Systems incorporating Bloom filters can operate under such a partial-adaptivity model in many real-world scenarios, including content caching (e.g., as in content delivery networks)

and database systems. For example, LSM (log-structured merge) Tree stores, including Google’s LevelDB [11] and Facebook’s RocksDB [17], use Bloom filters to reduce read times [6]. These stores can receive queries from both malicious and non-malicious users, which captures the fact that \mathcal{A} can observe the output of queries made by others while it can choose the rest.

In the partial adaptivity game **PAGame**, as in **FAGame**, \mathcal{A} ’s goal is to produce a previously unseen false positive. However, this time, \mathcal{A} cannot choose all the queries; it can only choose a fraction α of them, while the remaining $(1 - \alpha)t$ queries are uniformly sampled at random from \mathcal{D} . \mathcal{A} , however, still observes the output of all queries. \mathcal{A} also has the freedom to choose the order in which adversarial queries are interleaved between non-adversarial queries. We show this notion first for CBFs, where c indicates \mathcal{A} ’s choice of whether to evaluate the adversarial query if $c = 1$, or a non-adversarial one if $c = 0$.

PAGame($\mathcal{A}, \alpha, t, \lambda$):

1. $S \leftarrow \mathcal{A}_1(1^\lambda)$ such that $S \subseteq \mathcal{D}$ and $|S| = n$.
2. $M \leftarrow \mathbf{B}_1(S)$, and set $i = \beta = 0$.
3. $(c, x_i) \leftarrow \mathcal{A}_2^{\mathbf{B}_1(\cdot)}(1^\lambda, S, i, \beta)$ — \mathcal{A}_2 can make any polynomial number of queries to \mathbf{B}_1 .
4. If $c = 1$ and $\beta < \alpha t$, give \mathcal{A}_2 the output of $\mathbf{B}_2(M, x_i)$, and set $\beta = \beta + 1$.
5. Otherwise, $x_i \leftarrow_s \mathcal{D}$ and give \mathcal{A}_2 the output of $\mathbf{B}_2(M, x_i)$.
6. Set $i = i + 1$. If $i < t$, go back to Step 3.
7. \mathcal{A}_2 outputs x^* . If $x^* \notin S \cup \{x_1, \dots, x_t\}$ and $\mathbf{B}_2(M, x^*) = 1$, return 1. Otherwise, return 0.

This security game can be modified to work for LBFs as follows.

LPAGame($\mathcal{A}, \alpha, t, \lambda$):

1. $S \leftarrow \mathcal{A}_1(1^\lambda)$ such that $S \subseteq \mathcal{D}$ and $|S| = n$.
2. $M \leftarrow \mathbf{B}_1(S)$, $\mathcal{J} \leftarrow \mathbf{B}_3(S)$, $(\mathcal{L}, \tau) \leftarrow \mathbf{B}_4(\mathcal{J})$, and set $i = \beta = 0$.
3. $(c, x_i) \leftarrow \mathcal{A}_2^{\mathbf{O}}(1^\lambda, S, i, \beta)$, where $\mathbf{O} = \{\mathbf{B}_1(\cdot), \mathbf{B}_3(\cdot), \mathbf{B}_4(\cdot)\}$ and \mathcal{A}_2 can make any polynomial number of queries to \mathbf{O} .
4. If $c = 1$ and $\beta < \alpha t$, give \mathcal{A}_2 the output of $\mathbf{B}_2(M, \mathcal{L}, \tau, x_i)$, and set $\beta = \beta + 1$.
5. Otherwise, $x_i \leftarrow_s \mathcal{D}$ and give \mathcal{A}_2 the output of $\mathbf{B}_2(M, \mathcal{L}, \tau, x_i)$.
6. Set $i = i + 1$. If $i < t$, go back to Step 3.
7. \mathcal{A}_2 outputs x^* . If $x^* \notin S \cup \{x_1, \dots, x_t\}$ and $\mathbf{B}_2(M, \mathcal{L}, \tau, x^*) = 1$, return 1. Otherwise, return 0.

Definition 2 still applies to these games with one change; now we say a Bloom filter is $(n, \alpha, t, \varepsilon)$ -secure to account for the additional parameter α .

3.4 Learned Bet-or-Pass Security

This section extends the CBF **BPGame** from Naor and Oved [21] to LBFs, denoted as **LBPGame** (the same expectation notion from Definition 3 applies here as well).

LBPGame(\mathcal{A}, t, λ):

1. $S \leftarrow \mathcal{A}_1(1^\lambda)$ such that $S \subseteq \mathfrak{D}$ and $|S| = n$.
2. $M \leftarrow \mathbf{B}_1(S)$, $\mathcal{T} \leftarrow \mathbf{B}_3(S)$, $(\mathcal{L}, \tau) \leftarrow \mathbf{B}_4(\mathcal{T})$.
3. $(b, x^*) \leftarrow \mathcal{A}_2^\mathcal{O}(1^\lambda, S)$ where $\mathcal{O} = \{\mathbf{B}_1(\cdot), \mathbf{B}_2(M, \mathcal{L}, \tau, \cdot), \mathbf{B}_3(\cdot), \mathbf{B}_4(\cdot)\}$. \mathcal{A}_2 can make at most t queries to \mathbf{B}_2 , and any polynomial number of queries to each of \mathbf{B}_1 , \mathbf{B}_3 , and \mathbf{B}_4 .
4. Return \mathcal{A} 's profit, $C_{\mathcal{A}}$, which is defined as

$$C_{\mathcal{A}} = \begin{cases} 0, & \text{if } b = 0 \\ \varepsilon^{-1}, & \text{if } b = 1 \text{ and } x^* \notin S \cup \{x_1, \dots, x_t\} \\ & \text{and } \mathbf{B}_2(M, \mathcal{L}, \tau, x^*) = 1 \\ -(1 - \varepsilon)^{-1}, & \text{otherwise} \end{cases}$$

3.5 Relationships Between Security Notions

We investigate relationships between the security games we defined so far. For clarity, we focus on LBF versions of all security games. Naor et al. [15, 21] investigate similar relationships for the CBF security notions they defined, including ABGame and BPGame.

We first explore connections between LFAGame and LPAGame. Notice that when $\alpha = 1$, LPAGame is equivalent to LFAGame. LFAGame, therefore, is the special case of LPAGame where all queries are adversarial. The converse relationship is false, meaning that LFAGame is stronger than LPAGame.

Theorem 1. For $\varepsilon \in (0, 1)$, $\alpha \in [0, 1]$ and $n, t \in \mathbb{N}$, we have:

$$(n, t, \varepsilon)\text{-security in LFAGame} \implies (n, \alpha, t, \varepsilon)\text{-security in LPAGame}$$

$$(n, \alpha, t, \varepsilon)\text{-security in LPAGame} \not\Rightarrow (n, t, \varepsilon)\text{-security in LFAGame}$$

Proof. For the first relationship, fix a PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ and $t, \lambda \in \mathbb{N}$. Pick any $\alpha, \alpha' \in [0, 1]$ such that $\alpha' \geq \alpha$. Construct another adversary $\mathcal{A}' = (\mathcal{A}'_1, \mathcal{A}'_2)$ as follows. Let $\mathcal{A}'_1(1^\lambda, \alpha', t) = \mathcal{A}_1(1^\lambda, \alpha, t)$ and $\mathcal{A}'_2(1^\lambda, S, \alpha', t, \beta, i) = \mathcal{A}_2(1^\lambda, S, \alpha, t, \beta, i)$. This does not break the rules of LPAGame as still $\alpha' \in [0, 1]$. Since $\alpha' \geq \alpha$, the winning probability of \mathcal{A}' is at least as high as \mathcal{A} 's winning probability. When $\alpha' = 1$, we have $\text{LPAGame}(\mathcal{A}', 1, t, \lambda) = \text{LFAGame}(\mathcal{A}', t, \lambda)$, thus the result follows.

For the second relationship, let \mathbf{B} be an $(n, \alpha, t, \varepsilon)$ -secure LBF construction in LPAGame. Consider LFAGame with the same parameters (excluding α) as LPAGame. Also, let \mathbf{B}' be an alternative LBF construction such that the construction algorithm \mathbf{B}' is identical to \mathbf{B} , while the query algorithm \mathbf{B}'_2 differs from \mathbf{B}_2 as follows. Initially, $\mathbf{B}'_2(M, \mathcal{L}, \tau, \cdot) = \mathbf{B}_2(M, \mathcal{L}, \tau, \cdot)$. However, \mathbf{B}'_2 tracks the number of identical queries it receives. On receiving t identical queries, \mathbf{B}'_2 switches to always output 1 for all future queries. An adversary can win LFAGame with

non-negligible probability by having all its t queries over the same value x . At this point, \mathbf{B}'_2 will output 1 for any guess $x^* \neq x$ that the adversary outputs and wins the game. Therefore, \mathbf{B}' is not $(n, \alpha, t, \varepsilon)$ -secure in LFAGame . Now, consider an adversary \mathcal{A} in LPAGame . To win with non-negligible probability, \mathcal{A} must trigger \mathbf{B}'_2 to always output 1. \mathcal{A} can have at most $t - 1$ adversarial queries, and can choose all of them to be over the same value x . However, to trigger \mathbf{B}'_2 to always output 1, the final query, which is chosen uniformly randomly by the challenger, must also be x . Assuming the domain \mathfrak{D} is large, the probability of this happens is negligible. Therefore, \mathbf{B}' remains $(n, \alpha, t, \varepsilon)$ -secure in LPAGame . \square

We now compare LFAGame and LPAGame to LABGame and LBPGame .

Theorem 2. For $\varepsilon \in (0, 1)$, $\alpha \in [0, 1]$ and $n, t \in \mathbb{N}$, we have:

$$(n, t, \varepsilon)\text{-security in LFAGame} \implies (n, t, \varepsilon)\text{-security in LABGame}$$

$$(n, \alpha, t, \varepsilon)\text{-security in LPAGame} \implies (n, \alpha t, \varepsilon)\text{-security in LABGame}$$

Proof. Fix a PPT adversary \mathcal{A} , and $t, \lambda \in \mathbb{N}$. $\text{LABGame}(\mathcal{A}, t, \lambda)$ is identical to $\text{LFAGame}(\mathcal{A}, t, \lambda)$ aside from the fact that \mathcal{A} does not have oracle access to $\mathbf{B}_1(\cdot)$ in LABGame . Hence, \mathcal{A} 's winning probability in $\text{LABGame}(\mathcal{A}, t, \lambda)$ cannot be greater than \mathcal{A} 's winning probability in $\text{LFAGame}(\mathcal{A}, t, \lambda)$. Following a similar argument, we get an implication between LPAGame and LABGame . \square

To disprove the converse, we introduce an (n, t, ε) -secure construction under LABGame and then modify it such that it remains secure under LABGame but not under LFAGame . We first recall the following theorem for CBFs by Naor and Yagev [20].

Theorem 3 (Naor-Yagev Theorem). Let \mathbf{B} be an (n, ε) -Bloom filter using m memory bits. If pseudorandom permutations (PRPs) exist, then for security parameter λ there exists a negligible function $\text{negl}(\cdot)$ and an $(n, \varepsilon + \text{negl}(\lambda))$ -strongly resilient Bloom filter in ABGame that uses $m' = m + \lambda$ bits of memory.¹

This theorem is proved for the following construction, which we denote as Naor-Yagev CBF, or NY-CBF. Run the initialization algorithm of a CBF with the set $S' = \{\text{PRP}_{\text{sk}}(x) : x \in S\}$ instead of S , where PRP_{sk} is a keyed PRP. Similarly, for an element $x \in \mathfrak{D}$, query the filter over $\text{PRP}_{\text{sk}}(x)$ instead of x . This new CBF construction uses $m + \lambda$ bits of memory and is $(n, \varepsilon + \text{negl}(\lambda))$ -secure for any $t \in \mathcal{O}(\text{poly}(n, \lambda))$ under ABGame . We modify this construction in the proof for the theorem below.

Theorem 4. For $\varepsilon \in (0, 1)$, $\alpha \in [0, 1]$, $n, t \in \mathbb{N}$, and $\delta \in (0, 1)$, we have:

$$(n, t, \varepsilon)\text{-security in LABGame} \not\Rightarrow (n, t, \delta)\text{-security in LFAGame}$$

$$(n, \alpha t, \varepsilon)\text{-security in LABGame} \not\Rightarrow (n, \alpha, t, \delta)\text{-security in LPAGame}$$

¹ Strongly resilient means being (n, t, ε) -secure under BPGame for any $t \in \mathcal{O}(\text{poly}(n, \lambda))$. For more details, see Definition 2.4 in [21].

Proof. Let \mathbf{B} be an NY-CBF that is (n, t, ε) -secure under \mathbf{ABGame} . Consider a construction \mathbf{B}' that is (n, t, ε) -secure under $\mathbf{LABGame}$. \mathbf{B}' replaces \mathbf{B} with standard (or conventional) LBF that uses \mathbf{B} as its backup CBF and has a trivial learning model that responds negative (i.e., $x \notin S$) to all queries $x \in \mathcal{D}$. Although contrived, \mathbf{B}' is a correct LBF by Definition 6. Consider a PPT adversary \mathcal{A} . Since the learned representation of \mathbf{B}' contains no information on the input set S and routes all queries to its backup CBF, which is a NY-CBF, \mathcal{A} in $\mathbf{LABGame}$ has no advantage over \mathcal{A} in \mathbf{ABGame} . Hence, \mathbf{B}' is also (n, t, ε) -secure in $\mathbf{LABGame}$.

Now consider a second construction \mathbf{B}'' that is identical to \mathbf{B}' but with one modification: instead of storing the internal representation M like \mathbf{B}' , \mathbf{B}'' stores the internal representation $M' = (M, \text{sk})$ where sk is the secret key of PRP. \mathbf{B}'' is still $(n, t, \delta = \varepsilon + \text{negl}(\lambda))$ -secure under $\mathbf{LABGame}$ as \mathcal{A} does not have access to the internal representation of the Bloom filter. However, \mathbf{B}'' is not $(n, t, \delta = \varepsilon + \text{negl}(\lambda))$ -secure under $\mathbf{LFAGame}$ or $\mathbf{LPAGame}$ where \mathcal{A} has oracle access to M' and can obtain the secret key sk . \square

Before we show our next result, we recall a theorem by Naor and Oved [21], in which they proved that security in \mathbf{ABGame} does not imply security in \mathbf{BPGame} . Their proof uses a counterexample construction that usually behaves like an (n, t, ε) -secure CBF in \mathbf{ABGame} but has a small probability of reaching an always-one state, i.e., the query algorithm always outputs 1 for any query.

Theorem 5 (Naor-Oved Theorem). Let $0 < \varepsilon < 1$ and $n \in \mathbb{N}$, then for any $0 < \delta < 1$, assuming the existence of one-way functions, there exists a non-trivial Bloom filter \mathbf{B} that is (n, ε) -strongly resilient under \mathbf{ABGame} and is not (n, δ) -strongly resilient under \mathbf{BPGame} .

Theorem 6. For $\varepsilon \in (0, 1)$, $n, t \in \mathbb{N}$, and $\delta \in (0, 1)$, we have:

$$(n, t, \varepsilon)\text{-security in } \mathbf{LABGame} \not\Rightarrow (n, t, \delta)\text{-security in } \mathbf{LBPGame}$$

$$(n, t, \varepsilon)\text{-security in } \mathbf{LBPGame} \implies (n, t, \varepsilon)\text{-security in } \mathbf{LFAGame} \text{ and } \mathbf{LABGame}$$

Proof. For the first relationship, let \mathbf{B} be the Naor-Oved CBF construction that is (n, t, ε) -secure under \mathbf{ABGame} but not under \mathbf{BPGame} . We demonstrated in Theorem 4 how to create a construction \mathbf{B}' that is (n, t, ε) -secure under $\mathbf{LABGame}$ by having a trivial learning model that routes all queries to the backup CBF. This backup CBF is \mathbf{B} , which is secure under the \mathbf{ABGame} , making the overall construction secure under the $\mathbf{LABGame}$. By Theorem 5, we know that \mathbf{B} is not (n, t, δ) -secure under \mathbf{BPGame} for any $\delta \in (0, 1)$. Since the learning model in \mathbf{B}' is trivial and adds no adversarial resilience, it follows that \mathbf{B}' is not (n, t, δ) -secure under $\mathbf{LBPGame}$ for any $\delta \in (0, 1)$.

For the second relationship, Naor and Oved [21] also prove that for CBFs (n, t, ε) -security under \mathbf{BPGame} implies (n, t, ε) -security under \mathbf{ABGame} . We use a proof technique similar to theirs. Let \mathbf{B} be a construction that is (n, t, ε) -secure

under **LBPGame**. Fix an **LBPGame** adversary \mathcal{A} that outputs a guess x^* . Let FP denote the event that x^* is a false positive. For construction \mathbf{B} , the expected profit of \mathcal{A} is:

$$\begin{aligned} \mathbb{E}[C_A] &= \frac{1}{\varepsilon} \Pr[\text{FP}] - \frac{1}{1-\varepsilon} \Pr[\neg\text{FP}] = \frac{1}{\varepsilon} \Pr[\text{FP}] - \frac{1}{1-\varepsilon} (1 - \Pr[\text{FP}]) \\ &= \left(\frac{1}{\varepsilon} + \frac{1}{1-\varepsilon} \right) \Pr[\text{FP}] - \frac{1}{1-\varepsilon} = \frac{1}{\varepsilon(1-\varepsilon)} \Pr[\text{FP}] - \frac{1}{1-\varepsilon} \end{aligned}$$

Using $\mathbb{E}[C_A] \leq \text{negl}(\lambda)$, we obtain

$$\frac{1}{\varepsilon(1-\varepsilon)} \Pr[\text{FP}] - \frac{1}{1-\varepsilon} \leq \text{negl}(\lambda)$$

Since $\varepsilon \in (0, 1)$, we have $\varepsilon(1-\varepsilon) \in (0, 1)$, and thus

$$\Pr[\text{FP}] \leq \varepsilon + \varepsilon(1-\varepsilon)\text{negl}(\lambda) \leq \varepsilon + \text{negl}(\lambda)$$

We have shown that the probability of x^* being a false positive is at most negligibly greater than ε . This is the condition needed for \mathbf{B} to be (n, t, ε) -secure under **LFAGame** and **LABGame**. \square

Finally, we show that security in **LFAGame** does not imply security in **LBPGame**. As mentioned before, Naor and Oved proved Theorem 5 using a CBF counterexample that has a small probability of reaching an always-one state. We extend this idea to LBFs and prove that security in **LFAGame** does not imply security in **LBPGame**. Our proof is simpler than Naor and Oved's proof for CBFs because we give the adversary an oracle access to the Bloom filter's internal representation.

Theorem 7. For $\varepsilon \in (0, 1)$, $n, t \in \mathbb{N}$, and $\delta \in (0, 1)$, we have:

$$(n, t, \varepsilon)\text{-security in LFAGame} \not\Rightarrow (n, t, \delta)\text{-security in LBPGame}$$

Proof. Fix an $\varepsilon' \in (0, \varepsilon)$ and let \mathbf{B} be an LBF construction that is (n, t, ε') -secure in **LFAGame** (we show the existence of such a construction in Theorem 10). Let \mathbf{B}' be a modified construction that behaves as follows. The construction algorithm \mathbf{B}'_1 flips a bit b with probability p of being 1. If $b = 1$, \mathbf{B}' always answers 1 to any query. Otherwise, \mathbf{B}' behaves identically to \mathbf{B} . Let M be the internal classical representation of \mathbf{B} . \mathbf{B}' stores $M' = (M, b)$ as its internal classical representation. We first show that \mathbf{B}' is (n, t, ε) -secure in **LFAGame**. An adversary \mathcal{A} in **LFAGame** is always required to output a guess x^* . Let FP denote the event that x^* is a false positive. Note that $\Pr[\text{FP} \mid b = 1] = 1$ while $\Pr[\text{FP} \mid b = 0]$ is same as the probability of \mathcal{A} winning **LFAGame** with construction \mathbf{B} , i.e., $\Pr[\text{FP} \mid b = 0] \leq \varepsilon' + \text{negl}(\lambda)$. Based on that, and by choosing $p = \frac{\varepsilon - \varepsilon'}{1 - \varepsilon'}$,

we have:

$$\begin{aligned}
 \Pr[\text{FP}] &= \Pr[\text{FP} \mid b = 1] \Pr[b = 1] + \Pr[\text{FP} \mid b = 0] \Pr[b = 0] \\
 &\leq p + (\varepsilon' + \text{negl}(\lambda))(1 - p) \\
 &\leq \varepsilon' + p(1 - \varepsilon') + \text{negl}(\lambda) \\
 &\leq \varepsilon' + \frac{\varepsilon - \varepsilon'}{1 - \varepsilon'}(1 - \varepsilon') + \text{negl}(\lambda) \\
 &\leq \varepsilon + \text{negl}(\lambda)
 \end{aligned}$$

Thus, \mathbf{B}' is (n, t, ε) -secure in LFAGame . All that is left to show is that \mathbf{B}' is not (n, t, ε) -secure in LBPGame . Consider an adversary \mathcal{A}' in LBPGame . Recall that \mathcal{A}' has oracle access to \mathbf{B}' 's internal classical representation M' . Once \mathbf{B}' is constructed, \mathcal{A}' can read b in M' to check whether \mathbf{B}' is in the always-one state. \mathcal{A}' chooses to bet only if $b = 1$. Thus, we have:

$$\begin{aligned}
 \mathbb{E}[C_{\mathcal{A}'}] &= \mathbb{E}[C_{\mathcal{A}'} \mid b = 1] \Pr[b = 1] + \underbrace{\mathbb{E}[C_{\mathcal{A}'} \mid b = 0] \Pr[b = 0]}_{0 \text{ since } \mathcal{A}' \text{ won't bet}} \\
 &= \mathbb{E}[C_{\mathcal{A}'} \mid b = 1] \Pr[b = 1] = \varepsilon^{-1}p \geq p
 \end{aligned}$$

Therefore, $\mathbb{E}[C_{\mathcal{A}'}]$ is not negligible, violating the (n, t, ε) -security in LBPGame . \square

Figure 2 summarizes the relationships that we proved in this section.

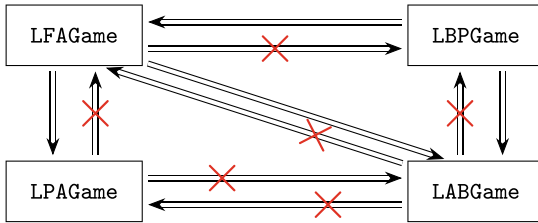


Fig. 2. Security notion implications.

4 Our Constructions

We propose two adaptively-secure LBF constructions; PRP-LBF and Cuckoo-LBF. In both of these constructions, we employ a Partitioned LBF [30] with a partition of cardinality 2. PRP-LBF combines partitioning with PRPs, while Cuckoo-LBF combines partitioning with Cuckoo hashing and pseudorandom functions (PRFs).

Construction 1: PRP-LBF. As shown in Fig. 3, in PRP-LBF \mathbf{B} , the learning model \mathcal{L} and threshold τ are used to partition $S \subseteq \mathcal{D}$ into two sets: $S_1 =$

$\{x \in S \mid \mathcal{L}(x) \geq \tau\}$ and $S_2 = S \setminus S_1 = \{x \in S \mid \mathcal{L}(x) < \tau\}$. We then use PRP_{sk_A} and PRP_{sk_B} as bijections on sets S_1 and S_2 , respectively, to form sets $S_A = \{\text{PRP}_{\text{sk}_A}(x) : x \in S_1\}$ and $S_B = \{\text{PRP}_{\text{sk}_B}(x) : x \in S_2\}$. PRP-LBF has two backup CBFs, \mathbf{B}_A that stores S_A and \mathbf{B}_B that stores S_B . To query PRP-LBF over an element $x \in \mathcal{D}$, we first evaluate the learning model over x . If $\mathcal{L}(x) \geq \tau$, then we compute $y = \text{PRP}_{\text{sk}_A}(x)$ and pass y to the query algorithm of \mathbf{B}_A to obtain an answer for the membership query on whether $x \in S$. On the other hand, if $\mathcal{L}(x) < \tau$, then we repeat the same steps but while using PRP_{sk_B} and the query algorithm of \mathbf{B}_B .

More formally, PRP-LBF is a data structure with six components $\mathbf{P} = (\mathbf{B}_A, \mathbf{B}_B, \mathbf{B}_3, \mathbf{B}_4, \text{PRP}_{\text{sk}_A}, \text{PRP}_{\text{sk}_B})$. $\mathbf{B}_A = (\mathbf{B}_{A_1}, \mathbf{B}_{A_2})$ and $\mathbf{B}_B = (\mathbf{B}_{B_1}, \mathbf{B}_{B_2})$ are backup CBFs (act like NY-CBF). \mathbf{B}_3 is a randomized dataset construction algorithm that on input S constructs a training dataset \mathcal{T} for S . \mathbf{B}_4 is a randomized learning model construction algorithm that on input \mathcal{T} outputs a learning model \mathcal{L} and a threshold $\tau \in [0, 1]$. PRP_{sk_A} and PRP_{sk_B} are pseudorandom permutations with secret keys sk_A and sk_B . The internal representation of S consists of:

1. M_A , the representation of $S_A = \{\text{PRP}_{\text{sk}_A}(x) : x \in S \mid \mathcal{L}(x) \geq \tau\}$ stored by backup CBF \mathbf{B}_A .
2. M_B , the representation of $S_B = \{\text{PRP}_{\text{sk}_B}(x) : x \in S \mid \mathcal{L}(x) < \tau\}$ stored by backup CBF \mathbf{B}_B .
3. (\mathcal{L}, τ) , the learning model and the threshold.

The query algorithm for PRP-LBF is $\mathbf{B}_2(M_A, M_B, \mathcal{L}, \tau, x) = (\mathcal{L}(x) \geq \tau \wedge \mathbf{B}_{A_2}(M_A, \text{PRP}_{\text{sk}_A}(x)) = 1) \vee (\mathcal{L}(x) < \tau \wedge \mathbf{B}_{B_2}(M_B, \text{PRP}_{\text{sk}_B}(x)) = 1)$. Similar to prior work [8, 20, 21], we assume that the internal state available to the adversary does **not** include the PRP secret keys, which are held securely.

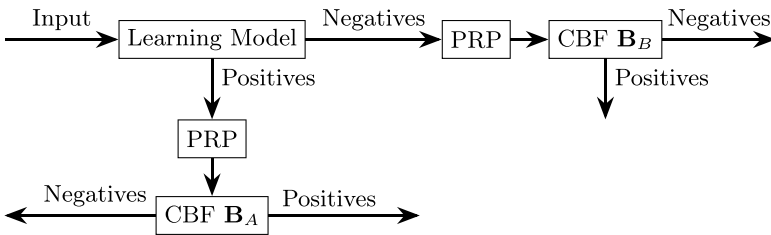


Fig. 3. The PRP-LBF construction.

Construction 2: Cuckoo-LBF. This construction, shown in Fig. 4, combines partitioning with Cuckoo hashing and PRFs. We first review prior constructions that also used Cuckoo hashing, which form the basis for ours. Naor et al. [15, 21] present a CBF construction, that is provably secure under BPGame , based on a prior Cuckoo hashing-based construction by Naor and Yaguev [20]. We denote

this construction Naor-Oved-Yogev Cuckoo filter or simply NOY-Cuckoo Filter. NOY-Cuckoo Filter is similar to a CBF variant called a Cuckoo filter [7]. For a set $S \subseteq \mathfrak{D}$ of size n , NOY-Cuckoo Filter $\mathbf{B} = (\mathbf{B}_1, \mathbf{B}_2)$ stores S using two tables Z_1, Z_2 , each with $n_c = \mathcal{O}(n)$ cells. Each table has a corresponding hash function, denoted as $h_1, h_2 : \mathfrak{D} \rightarrow [n_c]$, respectively. There is a fingerprint function, namely, a keyed pseudorandom function $\text{PRF}_{\text{sk}} : \mathfrak{D} \times \{1, 2\} \rightarrow \{0, 1\}^r$, where $r \in \mathcal{O}(\log \frac{1}{\varepsilon})$ for target FPR ε . The NOY-Cuckoo Filter works as follows:

- Construction algorithm \mathbf{B}_1 : Stores S in a Cuckoo hashing dictionary [22], where an element x is stored in either $Z_1[h_1(x)]$ or $Z_2[h_2(x)]$. To save space, the PRF output is stored instead of x . In particular, if x is to be stored in Z_1 , we store $y_1 = \text{PRF}_{\text{sk}}(x, 1)$. Otherwise, we store $y_2 = \text{PRF}_{\text{sk}}(x, 2)$ in Z_2 .
- Query algorithm \mathbf{B}_2 : To query $x \in \mathfrak{D}$, we compare $\text{PRF}_{\text{sk}}(x, 1)$ with $Z_1[h_1(x)]$ and $\text{PRF}_{\text{sk}}(x, 1)$ with $Z_2[h_2(x)]$. If either match occurs, we return 1.

NOY-Cuckoo Filter achieves the completeness and soundness properties from Definition 1. However, correctness is guaranteed only if the filter parameters are tuned carefully. Boskov et al. [2] show empirically that false negatives occur even in state-of-the-art Cuckoo filter implementations.

We combine NOY-Cuckoo Filter with the partitioning strategy we used for PRP-LBF to obtain Cuckoo-LBF. Formally, Cuckoo-LBF is a data structure with seven components $\mathbf{C} = (\mathcal{Z}, \mathcal{H}, \mathbf{B}_2, \mathbf{B}_3, \mathbf{B}_4, \text{PRF}_{\text{sk}_A}, \text{PRF}_{\text{sk}_B})$. $\mathcal{Z} = \{Z_1, \dots, Z_4\}$ is a set of 4 tables, and $\mathcal{H} = \{h_1, \dots, h_4\}$ is a set of 4 hash functions. $\mathbf{B}_2, \mathbf{B}_3, \mathbf{B}_4$ are polynomial time algorithms. \mathbf{B}_2 is a query algorithm, \mathbf{B}_3 is a randomized dataset construction algorithm, and \mathbf{B}_4 is a randomized learning model construction algorithm (with the same description as before). PRF_{sk_A} and PRF_{sk_B} are PRFs with secret keys sk_A and sk_B , respectively. The internal representation of a Cuckoo-LBF storing a set S consists of the following:

1. M_A , the representation of $S_A = \{x \in S \mid \mathcal{L}(x) \geq \tau\}$ stored by NOY-Cuckoo Filter \mathbf{B}_A , including tables Z_1, Z_2 , hash functions h_1, h_2 , and PRF_{sk_A} .
2. M_B , the representation of $S_B = \{x \in S \mid \mathcal{L}(x) < \tau\}$ stored by NOY-Cuckoo Filter \mathbf{B}_B , including tables Z_3, Z_4 , hash functions h_3, h_4 , and PRF_{sk_B} .
3. (\mathcal{L}, τ) , the learning model and the threshold.

The query algorithm \mathbf{B}_2 for a Cuckoo-LBF is $\mathbf{B}_2(M_A, M_B, \mathcal{L}, \tau, x) = (\mathcal{L}(x) \geq \tau \wedge \mathbf{B}_{A_2}(M_A, x) = 1) \vee (\mathcal{L}(x) < \tau \wedge \mathbf{B}_{B_2}(M_B, x) = 1)$ where \mathbf{B}_{A_2} and \mathbf{B}_{B_2} are query algorithms for the NOY-Cuckoo Filters \mathbf{B}_A and \mathbf{B}_B , respectively. As before, we assume that the internal state available to the adversary does **not** include the PRF secret keys.

False Positive Rate Analysis. We analyze FPR of both constructions in non-adversarial settings. We show that for PRP-LBF \mathbf{P} ; the same analysis and results hold for Cuckoo-LBF \mathbf{C} as both constructions use the same partitioning strategy. We also note that in our analysis, FPR refers to the probability of the event that some input $x \in \mathfrak{D}$ is a FP. Thus, in this and other sections that analyze FPR, we compute the probability of this event. A false positive (FP) for a query x happens in PRP-LBF if any of the following holds (see Fig. 5):

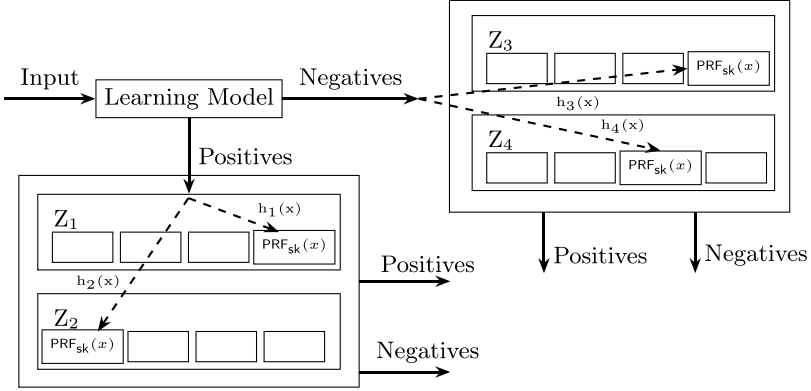


Fig. 4. The Cuckoo-LBF construction.

1. x generates a FP in \mathcal{L} and a FP in \mathbf{B}_A .
2. x generates a TN (true negative) in \mathcal{L} and a FP in \mathbf{B}_B .

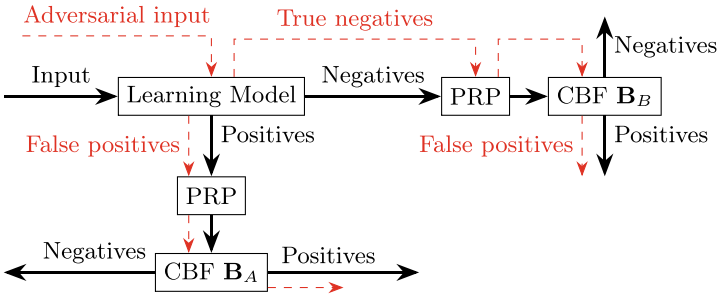


Fig. 5. To generate a false positive in a PRP-LBF, an adversary \mathcal{A} must either generate a false positive in the learning model and direct its query through backup CBF \mathbf{B}_A , or generate a true negative in the learning model and direct its query through backup CBF \mathbf{B}_B .

For $x \leftarrow_{\mathcal{S}} \mathcal{D}$,² let $\text{FP}(x, S, m)$ denote the event that x is a false positive in a CBF that encodes the set S with memory budget m , $\text{FP}_L(x, S, \mathcal{T}, m)$ denote the event that x is a false positive in a learning model \mathcal{L} generated using the training dataset \mathcal{T} (where this dataset is generated based on S) with memory budget m , and $\text{TN}_L(x, S, \mathcal{T}, m)$ denote the event that x is a true negative in a learning model \mathcal{L} generated using the training dataset \mathcal{T} (also based on the set S) with memory budget m . We assume that the correctness probability of

² The quantification of FPR here, or simply the probability a given input is FP, is for non-adversarial queries, hence, we have $x \leftarrow_{\mathcal{S}} \mathcal{D}$.

the learning model is independent of that of the backup CBFs. In particular, we assume that for any m, m' , we have:

$$\Pr[\text{FP}_L(x, S, \mathcal{T}, m) \cap \text{FP}(x, S, m')] = \Pr[\text{FP}_L(x, S, \mathcal{T}, m)] \Pr[\text{FP}(x, S, m')]$$

$$\Pr[\text{TN}_L(x, S, \mathcal{T}, m) \cap \text{FP}(x, S, m')] = \Pr[\text{TN}_L(x, S, \mathcal{T}, m)] \Pr[\text{FP}(x, S, m')]$$

Consider a total memory budget of m_T . The memory allocation of a PRP-LBF from m_T is assigned as follows: m_L bits are for the learning model \mathcal{L} , m_A bits are for backup CBF \mathbf{B}_A , m_B bits are for backup CBF \mathbf{B}_B , 2λ bits are for the PRP keys. Thus, we have $m_T \geq m_L + m_A + m_B + 2\lambda$. Based on that, we have the following theorem.

Theorem 8. Let $x \leftarrow_s \mathfrak{D}$, for any memory budget $m_T \in \mathbb{N}$, set $S \subseteq \mathfrak{D}$, let FP_{LBF} denote the event that x is a false positive in PRP-LBF (respectively, Cuckoo-LBF) encoding S . The probability of this event (in non-adversarial settings) is:

$$\begin{aligned} \Pr[\text{FP}_{LBF}(x, S, \mathcal{T}, m_T)] &= \Pr[\text{FP}_L(x, S, \mathcal{T}, m_L)] \Pr[\text{FP}(x, S_A, m_A)] \\ &\quad + \Pr[\text{TN}_L(x, S, \mathcal{T}, m_L)] \Pr[\text{FP}(x, S_B, m_B)] \end{aligned}$$

where \mathcal{T} and \mathcal{L} are the training dataset and learning model, respectively, corresponding to S , $S_A = \{x \in S \mid \mathcal{L}(x) \geq \tau\}$, $S_B = \{x \in S \mid \mathcal{L}(x) < \tau\}$, and the probability is taken over the random coins of \mathcal{L} and the backup CBFs used in PRP-LBF (respectively, Cuckoo-LBF).

Proof. Based on the design of PRP-LBF (and Cuckoo-LBF), it follows that a resulting FP must either be a FP in the backup CBF encoding S_A or in the backup CBF encoding S_B . A FP in the former must also be a FP in \mathcal{L} , while a FP in the latter must be a TN in \mathcal{L} . \square

5 Security and Performance Analysis

In this section, we formally prove security of our LBF constructions. We show that PRP-LBF is secure under LABGame , LFAGame , and LPAGame , and that Cuckoo-LBF is secure under the LBPGame and LABGame . We leave studying whether Cuckoo-LBF is secure under LFAGame and LPAGame or introducing a construction that is secure against all LBF games as a future work direction.

We also analyze FPR (achieved for a given memory budget) of our constructions under these adversarial settings. Our results show that our LBF constructions achieve strong security guarantees while maintaining competitive performance (in terms of FPR achieved for a given memory budget).

5.1 Fully Adaptive Adversary

This section establishes the security of PRP-LBF against fully adaptive adversaries. We first prove that NY-CBF maintains its security guarantees even when facing a fully adaptive adversary who can access the filter's internal representation. Building on this result, we then prove that PRP-LBF preserves these security properties in the LBF context.

Theorem 9. Let \mathbf{B} be an (n, ε) NY-CBF. Assuming PRPs exist, then for a security parameter λ there exists a negligible function $\text{negl}(\cdot)$ such that \mathbf{B} is $(n, t, \varepsilon + \text{negl}(\lambda))$ -secure under \mathbf{FAGame} for any $t \in \mathcal{O}(\text{poly}(n, \lambda))$.

Proof. By Theorem 3, we know that NY-CBF is a correct and sound CBF that is $(n, t, \varepsilon + \text{negl}(\lambda))$ -secure for any $t \in \mathcal{O}(\text{poly}(n, \lambda))$ under \mathbf{ABGame} . All that is left to show is that the additional oracle access to $\mathbf{O} = \{\mathbf{B}_1(\cdot), \mathbf{B}_2(M, \cdot)\}$ in \mathbf{FAGame} does not allow a PPT adversary \mathcal{A} to win in \mathbf{FAGame} with non-negligible probability. We can prove this using a hybrid argument. Let \mathbf{B} be a NY-CBF.

Hybrid Game $\mathbf{H0Game}$: This is the original game with \mathbf{B} .

Hybrid Game $\mathbf{H1Game}$: Let \mathbf{B}' be the same construction as \mathbf{B} but with PRP_{sk} replaced with a truly random permutation π .

Since PRP is a secure PRP, i.e., indistinguishable from a truly random permutation, thus \mathcal{A} cannot distinguish between $\mathbf{H0Game}$ and $\mathbf{H1Game}$. In $\mathbf{H1Game}$, to \mathcal{A} , the representation $\mathbf{B}'_1(\pi(S)) = M_\pi$ is indistinguishable from a representation constructed from a random set. So \mathcal{A} cannot gain any information about set S from $\mathbf{B}'_1(\pi(S))$ and \mathcal{A} 's view is identical to that in \mathbf{ABGame} . Hence, $\Pr[\mathcal{A} \text{ wins } \mathbf{H1Game}] \leq \Pr[\mathcal{A} \text{ wins } \mathbf{ABGame}] = \varepsilon + \text{negl}(\lambda)$.

Now, in the original construction \mathbf{B} that uses PRP_{sk} , assuming PRPs exist, we have $\Pr[\mathcal{A} \text{ wins } \mathbf{FAGame}] = \Pr[\mathcal{A} \text{ wins } \mathbf{H1Game}] + \text{negl}(\lambda) = \varepsilon + \text{negl}(\lambda)$. Therefore, NY-CBF is $(n, t, \varepsilon + \text{negl}(\lambda))$ -secure under \mathbf{FAGame} . \square

The following theorem shows that a PRP-LBF is $(n, t, \varepsilon + \text{negl}(\lambda))$ -secure under $\mathbf{LFAGame}$. To prove this result, we first prove that the differences between a PRP-LBF and a standard LBF construction still allow a PRP-LBF to be a correct $(n, \tau, \varepsilon, \varepsilon_p, \varepsilon_n)$ -LBF. Put differently, if there exists a standard LBF construction for any set $S \subseteq \mathcal{D}$ of cardinality n that satisfies the properties of an $(n, \tau, \varepsilon, \varepsilon_p, \varepsilon_n)$ -LBF, then there also exists a PRP-LBF construction that satisfies those properties. We prove security by doing a case analysis that reduces the security of PRP-LBF under $\mathbf{LFAGame}$ to the security of NY-CBF under \mathbf{FAGame} .

Theorem 10. Let \mathbf{B} be a standard construction for an $(n, \varepsilon, \varepsilon_p, \varepsilon_n)$ -LBF that uses m bits of memory out of which m_C bits are used for the backup CBF. Assuming PRPs exist, then for security parameter λ there exists a negligible function $\text{negl}(\cdot)$ and a PRP-LBF, $\mathbf{B}_{\text{sk}_A, \text{sk}_B}$, that is $(n, t, \varepsilon + \text{negl}(\lambda))$ -secure for any $t \in \mathcal{O}(\text{poly}(n, \lambda))$ under $\mathbf{LFAGame}$, and uses $m' = m + m_C + 2\lambda$ bits of memory.

Proof. Standard LBF \mathbf{B} can be transformed into a PRP-LBF \mathbf{P} as follows. Choose random secret keys $\mathbf{sk}_A, \mathbf{sk}_B \in \{0, 1\}^\lambda$ and use 2λ bits of extra memory to store them. Use the memory budget of \mathbf{B} 's backup CBF to construct backup CBF \mathbf{B}_A . Use m_C extra bits to construct backup CBF \mathbf{B}_B . Keep the learning model \mathcal{L} as is. \mathbf{P} 's completeness follows from \mathbf{B}_B 's completeness and the fact that for any x such that $\mathcal{L}(x) < \tau$, \mathbf{P} will return $x \notin S$ if and only if the query algorithm of \mathbf{B}_B outputs 0. The soundness of $\mathbf{B}_{\mathbf{sk}_A, \mathbf{sk}_B}$ follows from the soundness of \mathbf{B}_A and \mathbf{B}_B , and \mathbf{P} 's learning model soundness follows from \mathcal{L} 's soundness.

Consider a false positive (FP), i.e., an $x \notin S$ for which $\mathbf{B}_{\mathbf{sk}_A, \mathbf{sk}_B}$ returns 1. This occurs in two cases:

- Case 1: $\mathcal{L}(x) \geq \tau$ and \mathbf{B}_A returns 1, i.e., $\mathbf{B}_{A2}(M_A, \text{PRP}_{\mathbf{sk}_A}(x)) = 1$.
- Case 2: $\mathcal{L}(x) < \tau$ and \mathbf{B}_B returns 1, i.e., $\mathbf{B}_{B2}(M_B, \text{PRP}_{\mathbf{sk}_B}(x)) = 1$.

In both cases, for adversary \mathcal{A} to induce a FP in \mathbf{P} , it must induce a FP either in backup CBF \mathbf{B}_A or in backup CBF \mathbf{B}_B . \mathbf{B}_A and \mathbf{B}_B are NY-CBFs and, by Theorem 9, are $(n, t, \epsilon + \text{negl}(\lambda))$ -secure under FAGame . Therefore, \mathbf{P} is $(n, t, \epsilon + \text{negl}(\lambda))$ -secure under LFAGame . \square

We now discuss how PRP-LBF mitigate concrete attacks discussed in the literature, namely, the two attacks on LBFs by Reviriego et al. [23] (which we refer to as opaque-box and clear-box attacks), as well as a general poisoning attack on learned index structures introduced by Kornaropoulos et al. [13].

Opaque-Box Attack. The opaque-box adversarial model is similar to LABGame as both allow \mathcal{A} to query the LBF. However, in LABGame \mathcal{A} chooses the LBF's input set S , whereas in Reviriego et al.'s model, \mathcal{A} does not choose that. The opaque-box attack first tests elements until a false positive or a true positive is found. They it mutates the positive by changing a small fraction of the bits in the input to generate more false positives. The attack targets the learning model in standard LBFs by making it generate false positives without having the input reach the backup CBF. Unlike a standard LBF, PRP-LBF ensures both positive and negative queries are routed to a backup CBF that is (n, t, ϵ) -secure under ABGame . This ensures that the opaque-box attack will not induce a false positive in PRP-LBF with probability non-negligibly greater than ϵ .

Clear-Box Attack. The clear-box adversarial model is similar to LFAGame . With knowledge of the internal state of the LBF's learning model, \mathcal{A} generates mutations in a more sophisticated way. Reviriego et al. provide the example of a malicious URL dataset where \mathcal{A} may begin with a non-malicious URL and make changes such as removing the "s" in "https". Since PRP-LBF is (n, t, ϵ) -secure even when \mathcal{A} has access to oracle $\mathcal{O} = \{\mathbf{B}_1(\cdot), \mathbf{B}_3(\cdot), \mathbf{B}_4(\cdot)\}$, which reveals learning model state, PRP-LBF remains secure.

Poisoning Attacks. Kornaropoulos et al. [13] discuss an attack where \mathcal{A} poisons the learning model's training dataset by having maliciously-chosen inputs in this set. This poisoning attack modifies the training dataset, but not the queries sent to the LBF. The results of our LFAGame hold even if we let \mathcal{A} choose \mathcal{J} as long as

the challenger validates that \mathcal{T} satisfies Definition 4. To accommodate poisoning attacks, we can let \mathcal{A} choose a \mathcal{T} that is not validated by the challenger. Even in this relaxed version of **LFAGame**, PRP-LBF will prevent \mathcal{A} from inducing false positives with probability non-negligibly larger than ε . This is because our security guarantees do not rely on the learning model, but on the (n, t, ε) -secure backup CBFs, which do not use the training dataset \mathcal{T} .

Theorem 11. In **LFAGame**, for a PPT adversary \mathcal{A} that outputs a guess $x^* \in \mathfrak{D}$, the probability that x^* is a false positive in a PRP-LBF \mathbf{P} storing set $S \subseteq \mathfrak{D}$ with training dataset \mathcal{T} and learning model \mathcal{L} is:

$$\Pr[\text{FP}_{LBF}(x^*, S, \mathcal{T}, m_T)] \leq \max(\Pr[\text{FP}(x^*, S_A, m_A)], \Pr[\text{FP}(x^*, S_B, m_B)])$$

where $S_A = \{x \in S \mid \mathcal{L}(x) \geq \tau\}$, $S_B = S \setminus S_A = \{x \in S \mid \mathcal{L}(x) < \tau\}$, FP_{LBF} is the event denoting a false positive in \mathbf{P} , FP is the event denoting a false positive in NY-CBF, m_T , m_A , and m_B are the total memory of \mathbf{P} , memory used by backup CBF \mathbf{B}_A , and memory used by backup CBF \mathbf{B}_B , respectively, and the probability is taken over the random coins of \mathcal{A} and \mathbf{P} .

Proof. As established in Theorem 8, x^* can only induce a FP in \mathbf{P} if x^* also induces a FP or a TN in the learning model \mathcal{L} . Therefore, the probability of x^* inducing a FP in \mathbf{P} will be the probability of x^* inducing a FP in one of the backup CBFs, i.e., the probability will be $\Pr[\text{FP}(x^*, S_A, m_A)]$ or $\Pr[\text{FP}(x^*, S_B, m_B)]$. Thus, the upper bound above follows.³ \square

5.2 Partially Adaptive Adversary

Recall that Theorem 1 proves that any LBF that is (n, t, ε) -secure under **LFAGame** is (n, t, ε) -secure under **LPAGame**. Since PRP-LBF is (n, t, ε) -secure under **LFAGame**, as we proved in the previous section, it is also (n, t, ε) -secure under **LPAGame**. Note that in both **LFAGame** and **LPAGame**, a PPT adversary \mathcal{A} outputs a guess $x^* \in \mathfrak{D}$. The difference between **LFAGame** and **LPAGame** is the fraction α of \mathcal{A} 's initial exploratory query budget t . Therefore, for a guess $x^* \in \mathfrak{D}$ output by adversary in **LPAGame**, the probability of x^* inducing a FP in PRP-LBF is the same as the expression we derived for **LFAGame** in Theorem 11.

As discussed in Sect. 3.3, **LPAGame** is actually designed to capture is a *mixed workload* where a percentage of the queries are adversarial and the rest are non-adversarial. This is a more relevant scenario when it comes to analyzing real-world performance in terms of FPR for a given memory budget under a given workload. Thus, we analyze FPR over the t queries x_1, \dots, x_t in **LPAGame**, covering the αt adversarial queries and the $(1 - \alpha)t$ non-adversarial queries. For clarity, we refer to these t queries as workload queries to distinguish them from the guess x^* .

³ Note that if \mathcal{A} chooses x^* at random from \mathfrak{D} , then this reduces to the non-adversarial case analyzed in Theorem 8. The bound in that theorem also respects the bound stated in Theorem 11 above.

Without loss of generality, let α_P of the adversarial queries generate FPs in the learning model that go through backup CBF \mathbf{B}_A . Similarly, let α_N of the adversarial queries generate TNs in the learning model that go through backup CBF \mathbf{B}_B . Note that $\alpha = \alpha_P + \alpha_N$. The adversary makes at most $\alpha_P t$ queries that generate FPs in the learning model and at most $\alpha_N t$ queries that generate TNs in the learning model.

Theorem 12. In LPAGame, for a workload query $x_i \in \mathfrak{D}$, the probability that x_i is a false positive in a PRP-LBF \mathbf{P} storing set $S \subseteq \mathfrak{D}$ with training dataset \mathcal{T} and learning model \mathcal{L} is

$$\alpha_P \Pr[\text{FP}(x_i, S_A, m_A)] + \alpha_N \Pr[\text{FP}(x_i, S_B, m_B)] \\ + (1 - \alpha_P - \alpha_N) \Pr[\text{FP}_{LBF}(x_i, S, \mathcal{T}, m_T)]$$

where $S_A = \{x \in S \mid \mathcal{L}(x) \geq \tau\}$, $S_B = S \setminus S_A = \{x \in S \mid \mathcal{L}(x) < \tau\}$, $\text{FP}(\cdot)$ is the event denoting a false positive in a CBF, $\text{FP}_{LBF}(\cdot)$ is the event denoting a false positive in \mathbf{P} , α_P is the fraction of t queries chosen by \mathcal{A} that induce false positives in \mathcal{L} , and α_N is the fraction of t queries chosen by \mathcal{A} that induce TNs in \mathcal{L} . The probability is taken over the random coins used by \mathcal{A} , \mathbf{P} , and the generation of the $(1 - \alpha_P - \alpha_N)t$ non-adversarial queries.

Proof. One of the following cases holds for any query x_i among the t workload queries in LPAGame.

- Case 1: x_i is not adversary-generated. Therefore, as established by Theorem 8, these have $\Pr[\text{FP}_{LBF}(x_i, S, \mathcal{T}, m_T)]$ to be FP. There are $(1 - \alpha_P - \alpha_N)t$ such queries.
- Case 2: x_i is adversary-generated and generates a FP in the learning model $\mathcal{L}(x_i)$. Since $\mathcal{L}(x_i)$ generating a FP and $\mathcal{L}(x_i)$ generating a TN are mutually exclusive events, the probability of x_i inducing a FP in \mathbf{P} is just the probability of x_i inducing a FP in backup CBF \mathbf{B}_A , i.e., $\Pr[\text{FP}(x_i, S_A, m_A)]$. There are $\alpha_P t$ such queries.
- Case 3: x_i is adversary-generated and generates a TN in $\mathcal{L}(x_i)$. Similar to case 2, we can derive the probability of x_i inducing a FP to be $\Pr[\text{FP}(x_i, S_B, m_B)]$. There are $\alpha_N t$ such queries.

The statement of the theorem follows by applying the law of total probability. \square

5.3 Bet-or-Pass Adversary

NY-CBF has only been shown to be secure under ABGame. Naor and Oved [21] provide compelling reasons for why NY-CBF may not be secure under BPGame. Whether it is possible to modify NY-CBF in a way that makes it secure under BPGame is an open problem. On the other hand, NOY-Cuckoo Filter is secure under BPGame, proved by Naor and Oved [21], and we recall this result below.

Theorem 13 (NOY Theorem). Assuming one-way functions exist, for any $n \in \mathbb{N}$, universe of size $n < u$, and $0 < \varepsilon < 1/2$, there exists a Bloom filter that is (n, ε) -strongly resilient in BPGame and uses $\mathcal{O}(n \log \frac{1}{\varepsilon} + \lambda)$ bits of memory. There exists a CBF construction \mathbf{B}' (which is NOY-Cuckoo Filter mentioned above) where for any constant $0 < \varepsilon < 1/2$, \mathbf{B}' is an (n, ε) -strongly resilient in BPGame and uses $\mathcal{O}(n \log \frac{1}{\varepsilon} + \lambda)$ bits of memory.

We now show that our Cuckoo-LBF construction is secure under LBPGame . We prove this using a case analysis of all the decisions available to an adversary \mathcal{A} in LBPGame . Our case analysis shows that all decision paths reduce the security of Cuckoo-LBF under LBPGame to the security of NOY-Cuckoo Filter under BPGame .

Theorem 14. Let \mathbf{B} be a standard $(n, \varepsilon, \varepsilon_p, \varepsilon_n)$ -LBF that uses m bits of memory out of which m_C bits are used for the backup CBF and m_L bits are used for the learning model, such that $m_C + m_L = m$. Assuming one-way functions exist, for a security parameter λ , any $n \in \mathbb{N}$, domain \mathfrak{D} such that $n < |\mathfrak{D}|$, and $0 < \varepsilon < 0.5$ there exists an LBF that is $(n, t, \text{negl}(\lambda))$ -secure under LBPGame for any $t \in \mathcal{O}(\text{poly}(n, \lambda))$ and uses $m' = m_L + \mathcal{O}(n \log \frac{1}{\varepsilon} + \lambda)$ bits of memory.

Proof. By Theorem 13, we know that NOY-Cuckoo Filter is $(n, t, \text{negl}(\lambda))$ -secure CBF under BPGame for any $t \in \mathcal{O}(\text{poly}(n, \lambda))$, and uses $\mathcal{O}(n \log \frac{1}{\varepsilon} + \lambda)$ bits of memory. Recall that in LBPGame , unlike BPGame , \mathcal{A} has oracle access to the filter construction algorithm $\mathbf{B}_1(\cdot)$ which returns the internal representation M . Let BPGamePlus be a modified version of BPGame where \mathcal{A} has oracle access to $\mathbf{B}_1(\cdot)$.

We first show that NOY-Cuckoo Filter \mathbf{B} (that uses PRF_{sk}) is $(n, t, \text{negl}(\lambda))$ -secure under BPGamePlus . We define a hybrid game H1Game in which PRF_{sk} is replaced by a true random function f . By the security of PRFs, this hybrid is indistinguishable from the original game that uses PRFs. We denote H1Game 's internal representation of NOY-Cuckoo Filter as M' . To \mathcal{A} , the representation M' is indistinguishable from a representation constructed from a random set. So \mathcal{A} cannot gain any information about the input set S from M' and \mathcal{A} 's view is identical to that in BPGame . Hence $\Pr[\mathcal{A} \text{ wins H1Game}] \leq \Pr[\mathcal{A} \text{ wins BPGame}] = \varepsilon + \text{negl}(\lambda)$. Now, in the original construction \mathbf{B} that uses PRF_{sk} , assuming PRFs exist, we have $\Pr[\mathcal{A} \text{ wins BPGamePlus}] \leq \Pr[\mathcal{A} \text{ wins H1Game}] + \text{negl}(\lambda) = \varepsilon + \text{negl}(\lambda)$. Therefore, NOY-Cuckoo Filter remains $(n, t, \text{negl}(\lambda))$ -secure under BPGamePlus for any $t \in \mathcal{O}(\text{poly}(n, \lambda))$.

Let \mathbf{C} denote the Cuckoo-LBF construction with two backup NOY-Cuckoo Filters \mathbf{B}_A and \mathbf{B}_B . \mathbf{C} 's completeness follows from \mathbf{B}_B 's completeness and the fact that any x such that $\mathcal{L}(x) < \tau$ is declared to be not in S by \mathbf{B}_2 (which is the query algorithm of the Cuckoo-LBF) if and only if the query algorithm of \mathbf{B}_B also outputs 0. \mathbf{C} 's soundness follows directly from the soundness of \mathbf{B}_A and \mathbf{B}_B . \mathbf{C} 's learning model soundness follows from \mathcal{L} 's soundness. Hence, \mathbf{C} is a correct $(n, \tau, \varepsilon, \varepsilon_p, \varepsilon_n)$ -LBF.

Now, we show that the security of Cuckoo-LBF under LBPGame is reducible to the security of NOY-Cuckoo Filter under BPGamePlus . Let E_A and E_B be the events that the query goes through backup CBF \mathbf{B}_A and \mathbf{B}_B , respectively. Based on the construction of Cuckoo-LBF, E_A and E_B are mutually exclusive

events and that $\Pr[E_A \cup E_B] = 1$. Let C_T be the overall adversary profit. We denote by C_A and C_B the expected adversary profit from queries that go to backup CBFs \mathbf{B}_A and \mathbf{B}_B , respectively. Based on that, we have:

$$\mathbb{E}[C_T] = \mathbb{E}[C_A | E_A] \Pr[E_A] + \mathbb{E}[C_B | E_B] \Pr[E_B] \leq \mathbb{E}[C_A | E_A] + \mathbb{E}[C_B | E_B]$$

If the total profit C_T is non-negligible, it must be true that either $\mathbb{E}[C_A | E_A]$ or $\mathbb{E}[C_B | E_B]$ is non-negligible. However, since \mathbf{B}_A and \mathbf{B}_B are NOY-Cuckoo Filters, by Theorem 13 and our result regarding **BPGAMEPLUS** above, we know that $\mathbb{E}[C_A|E_A]$ and $\mathbb{E}[C_B|E_B]$ are negligible. Therefore, $\mathbb{E}[C_T]$ is negligible meaning that Cuckoo-LBF is $(n, t, \text{negl}(\lambda))$ -secure under **LBPGAME**. \square

The probability that the guess x^* that an adversary \mathcal{A} outputs be a false positive in Cuckoo-LBF is upper-bounded by the decision path where \mathcal{A} always chooses to bet, i.e., $b = 1$. In this decision path, the false positive probability of **LBPGAME** can be analyzed in a similar way as done for **LFAGAME**.

Theorem 15. In **LBPGAME**, for a PPT adversary \mathcal{A} that outputs a guess x^* , the probability that x^* is a false positive in a Cuckoo-LBF \mathbf{C} storing set S with model \mathcal{L} is $\leq \max(\Pr[\text{FP}(x^*, S_A, m_A)], \Pr[\text{FP}(x^*, S_B, m_B)])$ where $S_A = \{x \in S \mid \mathcal{L}(x) \geq \tau\}$, $S_B = S \setminus S_A = \{x \in S \mid \mathcal{L}(x) < \tau\}$, FP is the event denoting a false positive in NY-CBF, m_T , m_A , and m_B are the total memory of \mathbf{C} , memory used by backup CBF \mathbf{B}_A , and memory used by backup CBF \mathbf{B}_B , respectively, and the probability is taken over the random coins of \mathcal{A} and \mathbf{C} .

Proof. Since we assume \mathcal{A} always bets and never passes, it always outputs x^* to be tested whether it is a FP. A case analysis identical to Theorem 11 shows that the probability of x^* inducing a FP is either $\Pr[\text{FP}(x^*, S_A, m_A)]$ or $\Pr[\text{FP}(x^*, S_B, m_B)]$. The upper bound follows from the fact that x^* going through backup CBF \mathbf{B}_A and backup CBF \mathbf{B}_B are mutually exclusive events. \square

6 Evaluation

We focus on evaluating FPR vs. memory tradeoffs of our LBF constructions in comparison with known secure CBF constructions. In Sect. 6.1 we conduct a numerical analysis for that in **LPAGAME** and **LFAGAME** based on our FPR model covering a large number of parameters. Note that only PRP-LBF is included in this analysis, as proving the security of Cuckoo-LBF under **LPAGAME** and **LFAGAME** is left to future work. To emphasize the practicality of our constructions, Sect. 6.2 evaluates PRP-LBF and Cuckoo-LBF, and compares them with Naor et al.'s CBF constructions NY-CBF and NOY-Cuckoo Filter, for a common use-case in a non-adversarial setting.

6.1 Numerical Analysis

We show performance results for real-world scenarios in both the fully ($\alpha = 1$) and partially ($\alpha \leq 1$) adaptive adversarial models. The goal is to demonstrate

scenarios where using a PRP-LBF instead of a NY-CBF yields better FPR, under a given memory budget, while maintaining security guarantees. Broder et al. [3] show that the probability of $x \in \mathcal{D}$ being a FP in a CBF with n_b bits storing a set S using n_h hash functions, is $\Pr[\text{FP}(x, S, n_b)] = (1 - e^{-n_h|S|/n_b})^{n_h}$. We choose the number of hash functions n_h to be optimal $n_h = \ln 2 \cdot (n_b/|S|)$, as derived in [3].

Similar to [14, 18], we let the false positive probability of a learning model, $\Pr[\text{FP}_L]$, can be modeled as a fraction of that of a CBF storing set S for the same memory budget (i.e., the learning model has a better FPR than a CBF): $\Pr[\text{FP}_L(x, S, \mathcal{J}, m)] = c(1 - e^{-n_h|S|/n_b})^{n_h}$ where $c \leq 1$. Learning models have both FPs and TNs, and we note that the probability of an entry being a TN in the original set is constant as we assume the input set S does not change after construction. Let Q_N be the fraction of TN non-adversarial queries (where the number of adversarial queries is αN and so the number of non-adversarial queries is $|\mathcal{D}| - \alpha N$). Thus, we have

$$\Pr[\text{TN}_L(S, \mathcal{J}, n_b)] = (1 - \Pr[\text{FP}_L(S, \mathcal{J}, n_b)]) Q_N = (1 - c(1 - e^{-n_h|S|/n_b})^{n_h}) Q_N$$

We choose realistic values for our example from prior work on evaluating LBFs [14] on Google’s Transparency Report. We pick 2 Megabytes as our memory budget, m , chosen from the range of values in Fig. 10 of [14]. We choose the cardinality of the stored set, $|S|$, as 1.7 million based on the number of unique URLs in Google’s Transparency Report. Kraska et al. [14] demonstrates that an LBF with a memory budget of 2 Megabytes has 0.25 of the false positive rate of a CBF. Hence, we also use 0.25 as the value for c in our calculations. Following prior work [20], we use 128 bits as the size of our security parameter, λ . For the case of a PRP-LBF, we let the learning model take 1 Megabytes while dividing the remaining 1 Megabytes equally between backup CBFs \mathbf{B}_A and \mathbf{B}_B . The backup CBFs store S_A and S_B , respectively, in PRP-LBF. Our chosen values are summarized in Table 1. The full numerical analysis for our model uses 494 lines of C code and it is available at [12].

Table 1. Model parameters for PRP-LBF and NY-CBF.

Parameter		Value
m_T	Total memory budget	2 MB
m_L	Memory budget for the learning model	1 MB
m_A	Memory budget for backup CBF \mathbf{B}_A	0.5 MB
m_B	Memory budget for backup CBF \mathbf{B}_B	0.5 MB
n	Cardinality of stored set $S \subseteq \mathcal{D}$	1.7 Million
c	$\frac{\Pr[\text{FP}_L] \text{ of learning model}}{\Pr[\text{FP}] \text{ of CBF}}$ for same memory budget	0.25
λ	Security parameter	128 bits
Q_N	Fraction of true negative non-adversarial queries	0.5

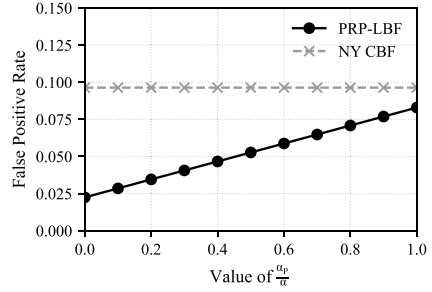
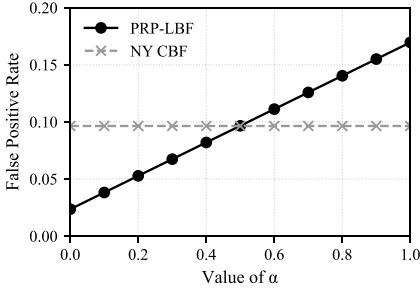
Varying the Fraction of Adversarial Queries. We take the fraction of adversarial queries α to be a variable ranging from 0 to 1. We assume a constant *adversarial strategy*, i.e., the fraction of adversarial queries that are FPs (so they go through backup CBF \mathbf{B}_A) vs. the fraction of adversarial queries that are TNs (so they go through backup CBF \mathbf{B}_B) is constant. In particular, we assume that adversary \mathcal{A} equally divides its queries between FPs and TNs, so $\frac{\alpha_P}{\alpha} = 0.5$ and Q_N is 0.5. As shown in Fig. 6a, we observe that a PRP-LBF outperforms a NY-CBF for the same memory budget when the fraction of adversarial queries is less than a certain cutoff of 0.5. So, as long as adversarial traffic is at most half of the total workload of an application, PRP-LBF will offer a lower FPR than NY-CBF. Note that for all our figures, the result for LFAGame is simply the $\alpha = 1$ point in the figure, whereas the entire spectrum of α values shows how FPR varies in the weaker LPAGame.

Varying the Adversarial Strategy. In Fig. 6b, we relax the assumption that the adversary divides their queries equally between FPs and TNs. We show results for all partitions of α between α_P and α_N . To see how the FPR of PRP-LBF is impacted, we vary the fraction of α assigned to α_P from 0 to 1. Here, 0 means all αN adversarial queries are TNs, and 1 means that all αN adversarial queries are FPs. In this framework, recall that our first calculation (Fig. 6a) sets this fraction to 0.5. The key observation here is that as \mathcal{A} uses more of its query budget directing traffic to the backup CBF that has the higher FPR, the performance of PRP-LBF degrades. Recall that the upper bound for the FPR of an adversarial query is the FPR of the “weaker” backup CBF. For brevity, Fig. 6b only shows results when $\alpha = 0.2$, i.e., 20% of the workload is adversarial. We note that we observed the same trend for other values of α , and hence, we do not include detailed results for that.

Varying the Dataset. In addition to Google’s Transparency Report, we also show results in Figs. 6c and 6d) for two other datasets evaluated in prior work on LBFs [5, 24, 30]. These two datasets are: Malicious URLs Dataset [25] that contains 223, 088 malicious and 428, 118 benign URLs, and EMBER Dataset [1] that contains 300, 000 malicious and 400, 000 benign files. We change the set’s cardinality values, n , in Table 1. We use the same values listed in the table for all other model parameters. The figures show that, similar to Google’s Transparency Report, also for these datasets FPR increases as the fraction of adversarial queries α increases.

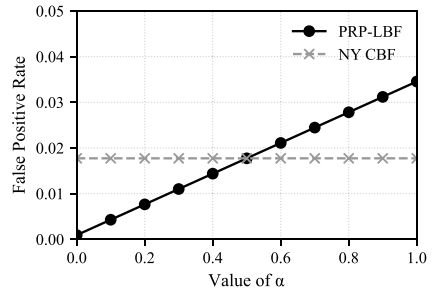
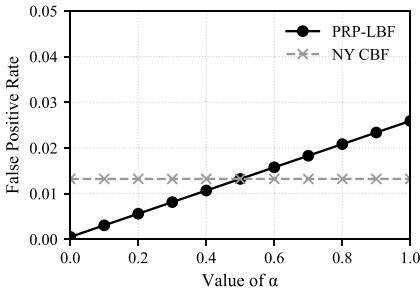
Varying the Fraction of True Negatives. Recall that in Fig. 6a, we set the fraction of non-adversarial queries that are TNs, Q_N , to be 0.5. Now we relax this assumption and show results for the entire range of values of $Q_N \in [0, 1]$ with α taking 4 values: 0.2, 0.3, 0.5, 1.0, such that each value partitioned equally between α_P and α_N . The results are shown in Fig. 7.

We observe that FPR increases with increasing Q_N but the rate of this increase, i.e., $\frac{\partial FPR}{\partial Q_N}$, decreases as α increases. This is due to the following reason. In Theorem 12, when $\alpha = \alpha_P + \alpha_N \rightarrow 1$, we have $(1 - \alpha_P - \alpha_N) \Pr[\text{FP}_{LBF}(x_i, S, \mathcal{J}, m_T)] \rightarrow 0$, so the overall FPR of the mixed workload



(a) Results for the Google Transparency Report as α varies with adversarial queries divided equally between FPs and TNs: $\alpha_P = \alpha_N$.

(b) Results for the Google Transparency Report with $\alpha = 0.2$ while varying the division of adversarial queries between FPs and TNs (i.e., α_P/α).



(c) Results for the Malicious URLs Dataset (same setting as Figure 6a).

(d) Results for the EMBER Dataset (same setting as Figure 6b).

Fig. 6. FPR comparison between PRP-LBF and NY-CBF while varying the fraction of adversarial queries α and the adversarial strategy α_P/α for various datasets. Results for the fully adaptive model are the $\alpha = 1$ points in the figures, whereas the entire spectrum of α values shows the results for the partially adaptive model.

is dominated by the FPR of the backup CBFs which does not depend on Q_N . On the other hand, when $\alpha \rightarrow 0$, the overall FPR of the mixed workload is dominated by FP_{LBF} which increases with increasing Q_N (see Theorem 8).

6.2 Use-Case Evaluation

We evaluate the performance of our constructions within the context of a use case to get a sense of how they would perform in practice. Historically, web browsers, including Google Chrome, used a Bloom filter [9] to store a set of Malicious URLs. In this design, whenever a user attempts to access a URL on the web browser, the browser queries the Bloom filter for the URL. If the Bloom filter says that the URL is in the Malicious URLs set, the web browser warns the

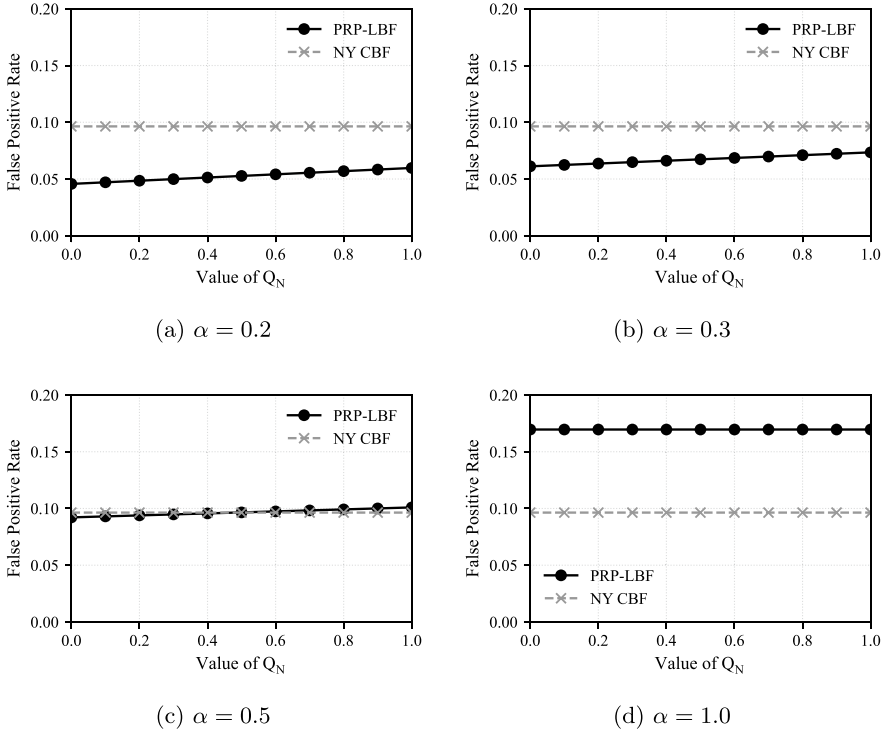


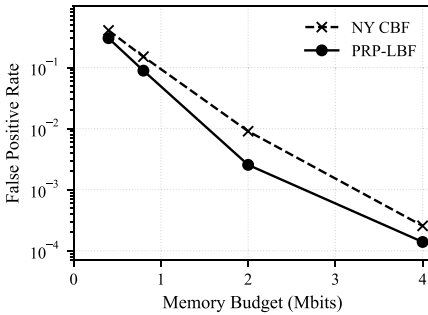
Fig. 7. FPR comparison between PRP-LBF and NY-CBF in the partially-adversarial model for the Google Transparency Report with $Q_N \in [0, 1]$.

user that they are accessing a potentially unsafe website. The malicious URLs use case has been studied by prior works on LBFs [24, 30]. Thus, we evaluate this use case using the same public Malicious URLs dataset [25] as prior work. This dataset contains around 223K malicious and around 428K benign URLs.

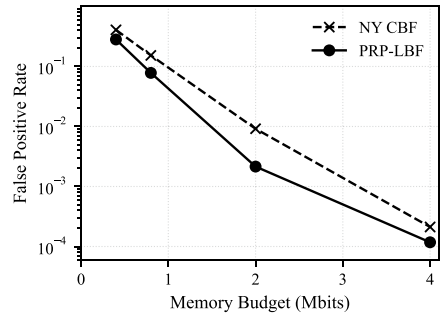
Implementation and Experimental Setup. We implemented PRP-LBF, Cuckoo-LBF, NY-CBF, and NOY-Cuckoo Filter in 903 lines of Python 3 code, which can be found in our open-source repository [12]. The implementation allows any PRP/PRF implementation to be plugged in for internal use. Similar to [20], we use AES to instantiate PRPs and PRFs in our implementation. In particular, we use AES in the ECB mode where the input size of the PRP/PRF is 128 bits (so one block for AES encryption). Our implementation uses the PyCryptoDome [26] library for these cryptographic primitives. Our implementation is also modular in a way that allows any machine learning model to be easily plugged in. We tested the correctness of our implementation on a broad range of common classification models, including the Random Forest model, Gaussian Naive Bayes, the Gradient Boosting Classifier, Support Vector Machine-based Classifiers, and Adaptive Boosting, using implementations provided by `scikit-learn`.

For our experiments, we use a set of 20 features to train the learning models, including URL length, whether the URL contains an IP address, whether the URL uses a shortening service, whether the URL is “abnormal”, digit count and letter count of the URL, and whether the URL contains special symbols. This set of features for the Malicious URLs Dataset is common in open-source learning models, and similar features have been used by prior work [30]. In the case of Cuckoo filter-based constructions, we use fingerprints of size 4 bits, 2 as the table size constant factor, and 5000 maximum eviction attempts in the Cuckoo hashing tables. For Cuckoo filter-based constructions, we also skip elements that cannot be inserted after the maximum eviction attempts have been reached.

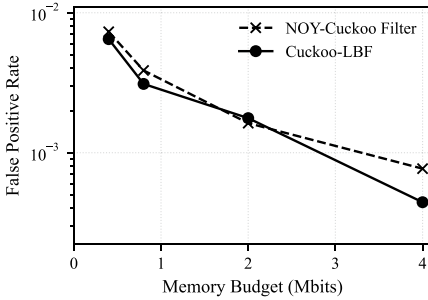
We calculate FPR of all constructions by uniformly randomly sampling 10% of URLs that are **not** malicious from the Malicious URLs Dataset and counting how many of them are returned as FPs by these constructions. The amount of memory the learning model uses is measured as the serialized size in bytes of the trained classifier, using joblib [27]. After subtracting the memory used by the learning model from the memory budget, we divide the remaining memory budget equally between backup CBFs \mathbf{B}_A and \mathbf{B}_B in our constructions.



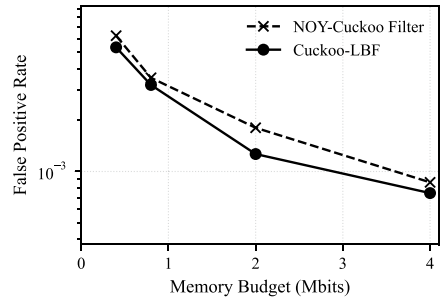
(a) Gaussian Naive-Bayes Classifier



(b) Linear Support Vector Classifier

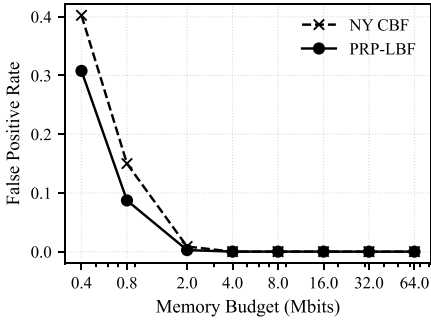


(c) Gaussian Naive-Bayes Classifier

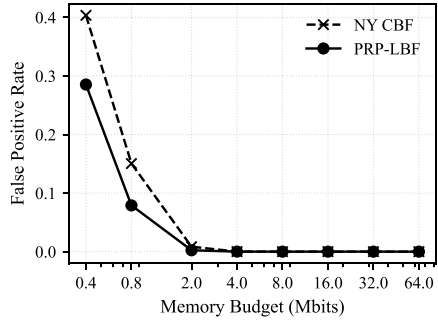


(d) Linear Support Vector Classifier

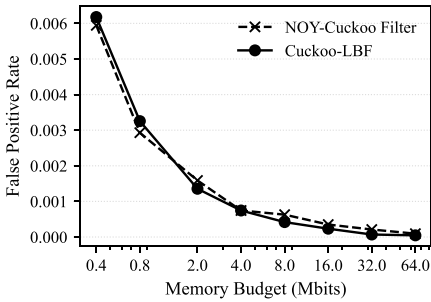
Fig. 8. FPR of our LBF constructions compared to NY-CBF and NOY-Cuckoo Filter, with varying memory budget and learning models for the Malicious URLs Dataset.



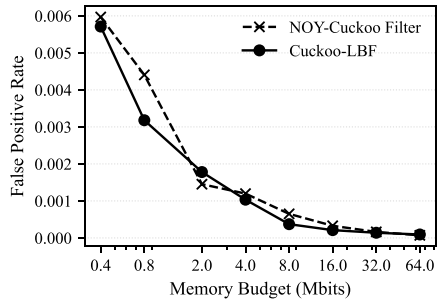
(a) Gaussian Naive-Bayes Classifier



(b) Linear Support Vector Classifier



(c) Gaussian Naive-Bayes Classifier



(d) Linear Support Vector Classifier

Fig. 9. FPR of our LBF constructions compared to NY-CBF and NOY-Cuckoo Filter with large memory budgets and different learning models (for the Malicious URLs Dataset).

Results. Figure 8a shows how FPR varies for NY-CBF and PRP-LBF as we modify the memory budget. This figure uses the Gaussian Naive-Bayes Classifier as the learning model. Figure 8b shows the same results using a Linear Support Vector Classifier as the learning model. Similarly, Fig. 8c shows how FPR varies for NOY-Cuckoo Filter and Cuckoo-LBF as we modify the memory budget available for the Gaussian Naive-Bayes Classifier and the Linear Support Vector Classifier as the learning model, respectively.

While our implementations are not focused on optimization, we see a consistent trend of our LBF constructions having lower FPRs than CBF constructions for the same memory budget. This is consistent with prior work [14] that shows a similar trend between non-adversarial LBF constructions and non-adversarial CBF constructions. An interesting outlier is the FPR of Cuckoo-LBF being slightly larger than the FPR of NOY-Cuckoo Filter in one of the data points of Fig. 8c

Large Memory Budgets. We explore how the trend of Fig. 8 continues as we keep increasing the memory budget to the point where it no longer becomes

a bottleneck for FPR. To better understand this, we conduct a second set of experiments over a much larger range of memory budgets with the results shown in Fig. 9. We observe that the trend from Fig. 8 continues. FPR for both our LBF constructions and the CBF constructions eventually approaches the same value as the memory budget increases.

Acknowledgments. The work of G.A. is supported by NSF Grant No. CNS-2226932.

Generative AI Disclosure. The paper was written directly by the authors. The student author used ChatGPT by OpenAI and Claude by Anthropic for feedback on technical writing and to edit and debug the TikZ and matplotlib figure code in the LaTeX manuscript. The artifacts of our work were primarily written and verified by the authors. The student author used Cursor IDE, which includes a generative AI code assistant, for the following tertiary coding tasks: unit-test generation, code refactoring, feedback on bug-fixing, feedback on anonymizing the artifact, and code reviews.

References

1. Anderson, H.S., Roth, P.: Ember: an open dataset for training static pe malware machine learning models. arXiv preprint [arXiv:1804.04637](https://arxiv.org/abs/1804.04637) (2018)
2. Boskov, N., Trachtenberg, A., Starobinski, D.: Birdwatching: false negatives in cuckoo filters. In: Proceedings of the Student Workshop, pp. 13–14 (2020)
3. Broder, A., Mitzenmacher, M.: Network applications of bloom filters: a survey. *Internet Math.* **1**(4), 485–509 (2004)
4. Clayton, D., Patton, C., Shrimpton, T.: Probabilistic data structures in adversarial environments. In: ACM SIGSAC Conference on Computer and Communications Security (CCS) (2019)
5. Dai, Z., Shrivastava, A.: 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 (NeurIPS) (2020)
6. Dayan, N., Athanassoulis, M., Idreos, S.: Monkey: optimal navigable key-value store. In: Proceedings of the 2017 ACM International Conference on Management of Data, pp. 79–94 (2017)
7. Fan, B., Andersen, D.G., Kaminsky, M., Mitzenmacher, M.D.: Cuckoo filter: practically better than bloom. In: Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies, pp. 75–88 (2014)
8. Filic, M., Paterson, K.G., Unnikrishnan, A., Virdia, F.: Adversarial correctness and privacy for probabilistic data structures. In: ACM SIGSAC Conference on Computer and Communications Security (CCS) (2022)
9. Gerbet, T., Kumar, A., Lauradoux, C.: On the (in) security of google safe browsing. INRIA ePrint (2014)
10. Gerbet, T., Kumar, A., Lauradoux, C.: The power of evil choices in bloom filters. In: IEEE/IFIP International Conference on Dependable Systems and Networks (2015)
11. Google: LevelDB Bloom Filter. <https://github.com/google/leveldb/blob/main/util/bloom.cc>, Accessed 04 May 2023
12. Open Source Implementation and Experiments (2025). <https://github.com/jadibourbaki/permuted-partitioned-lbf>

13. Kornaropoulos, E.M., Ren, S., Tamassia, R.: The price of tailoring the index to your data: Poisoning attacks on learned index structures. In: Proceedings of the 2022 International Conference on Management of Data, pp. 1331–1344 (2022)
14. Kraska, T., Beutel, A., Chi, E.H., Dean, J., Polyzotis, N.: The case for learned index structures. In: International Conference on Management of Data (2018)
15. Lotan, C., Naor, M.: Adversarially robust bloom filters: monotonicity and betting. *IACR Commun. Cryptol.* **2**(1) (2025)
16. Mangal, A., Roy, S., Akinapelli, S., Singh, A., Boulon, J.: How roblox reduces spark join query costs with machine learning optimized bloom filters. <https://corp.roblox.com/newsroom/2023/11/roblox-reduces-spark-join-query-costs-machine-learning-optimized-bloom-filters>
17. Meta: RocksDB Bloom Filter. https://github.com/facebook/rocksdb/blob/main/util/dynamic_bloom.h, Accessed 04 May 2023
18. Mitzenmacher, M.: A model for learned bloom filters and optimizing by sandwiching. *Adv. Neural Inform. Process. Syst.* **31** (2018)
19. Muthukrishnan, S.: Data streams: algorithms and applications. *Found. Trends Theor. Comput. Sci.* **1**(2), 117–236 (2005)
20. Naor, M., Eylon, Y.: Bloom filters in adversarial environments. *ACM Trans. Algorit. (TALG)* **15**(3), 1–30 (2019)
21. Naor, M., Oved, N.: Bet-or-pass: adversarially robust bloom filters. In: Theory of Cryptography Conference (2022)
22. Pagh, R., Rodler, F.F.: Cuckoo hashing. In: auf der Heide, F.M. (ed.) *ESA 2001*. LNCS, vol. 2161, pp. 121–133. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44676-1_10
23. Reviriego, P., Hernández, J.A., Dai, Z., Shrivastava, A.: Learned bloom filters in adversarial environments: A malicious url detection use-case. In: IEEE International Conference on High Performance Switching and Routing (HPSR). IEEE (2021)
24. Sato, A., Matsui, Y.: Fast partitioned learned bloom filter. In: International Conference on Neural Information Processing Systems (NeurIPS) (2024)
25. Siddharta, M.: Malicious URLs Dataset. <https://www.kaggle.com/datasets/sid321axn/malicious-urls-dataset>
26. Source, O.: PyCryptoDome library. <https://www.pycryptodome.org/>
27. Source, O.: Joblib (2025). <https://joblib.readthedocs.io>
28. Tarkoma, S., Rothenberg, C.E., Lagerspetz, E.: Theory and practice of bloom filters for distributed systems. *IEEE Commun. Surv. Tutorials* **14**(1), 131–155 (2012)
29. Tirmazi, H.: Lsm trees in adversarial environments. arXiv preprint [arXiv:2502.08832](https://arxiv.org/abs/2502.08832) (2025)
30. Vaidya, K., Knorr, E., Mitzenmacher, M., Kraska, T.: Partitioned learned bloom filters. In: International Conference on Learning Representations (2021)