Adversarial Correctness and Privacy for Probabilistic Data Structures

Mia Filić
ETH Zürich
Zürich, Switzerland
mia.filic@inf.ethz.ch

Kenneth G. Paterson
ETH Zürich
Zürich, Switzerland
kenny.paterson@inf.ethz.ch

Anupama Unnikrishnan
ETH Zürich
Zürich, Switzerland
anupama.unnikrishnan@inf.ethz.ch

Fernando Virdia*
Intel Labs
Zürich, Switzerland
fernando.virdia@intel.com

ABSTRACT

We study the security of Probabilistic Data Structures (PDS) for handling Approximate Membership Queries (AMQ); prominent examples of AMQ-PDS are Bloom and Cuckoo filters. AMQ-PDS are increasingly being deployed in environments where adversaries can gain benefit from carefully selecting inputs, for example to increase the false positive rate of an AMQ-PDS. They are also being used in settings where the inputs are sensitive and should remain private in the face of adversaries who can access an AMQ-PDS through an API or who can learn its internal state by compromising the system running the AMQ-PDS.

We develop simulation-based security definitions that speak to correctness and privacy of AMQ-PDS. Our definitions are general and apply to a broad range of adversarial settings. We use our definitions to analyse the behaviour of both Bloom filters and insertion-only Cuckoo filters. We show that these AMQ-PDS can be provably protected through replacement or composition of hash functions with keyed pseudorandom functions in their construction. We also examine the practical impact on storage size and computation of providing secure instances of Bloom and insertion-only Cuckoo filters.

CCS CONCEPTS

• **Security and privacy → Symmetric cryptography and hash functions**.

KEYWORDS

probabilistic data structures; simulation-based proofs; Bloom filters; Cuckoo filters

*The work of Virdia was carried out while employed by ETH Zürich.

CCS '22, November 7–11, 2022, Los Angeles, CA, USA

ACM ISBN 978-1-4503-9450-5/22/11…$15.00
https://doi.org/10.1145/3548606.3560621

ACM Reference Format:
Mia Filić, Kenneth G. Paterson, Anupama Unnikrishnan, and Fernando Virdia. 2022. Adversarial Correctness and Privacy for Probabilistic Data Structures. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS '22), November 7–11, 2022, Los Angeles, CA, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3548606.3560621

1 INTRODUCTION

Probabilistic data structures (PDS) are becoming ubiquitous in practice. They enjoy improved efficiency over exact data structures but this comes at the cost of them only giving approximate answers. Still, this is sufficient in many use-cases, for example when computing statistics on large data sets (e.g. finding the number of distinct items in the set [18], or moments of the frequency distribution of the set elements [1]), in answering set membership queries (e.g. to decide, in a storage-efficient manner, whether a particular data item has been encountered before [6]), or in identifying so-called *heavy hitters* in a set, that is, high frequency elements [13]. Often, these problems appear in the streaming setting, where memory is limited and items have to be processed in an online manner.

We refer to PDS that provide approximate answers to membership queries as AMQ-PDS; they are the focus of this paper. Prominent examples of AMQ-PDS include Bloom filters [6], Cuckoo filters [17], and Morton filters [9]. AMQ-PDS find use in applications ranging from certificate revocation systems (eg. CRLite) [26], database query speedup [34] and DNA sequence analysis [29].

As early as the 1990s it was recognised that simple data structures like hash tables could be manipulated into poor performance or leak information about their inputs [28]. Increasingly, PDS are being used in environments where the input may be adversarially chosen, see [12, 19, 33] for examples. However, PDS are not usually designed to work reliably in such settings, and their performance guarantees are typically proven for the case where the input distribution is benign. This disjunction can result in security vulnerabilities, for example, non-detection of network scanning attacks [33] or a Bloom filter reporting false positives on some targeted set of inputs [19]. PDS may also inadvertently leak information about their inputs in settings where it is desirable to keep those inputs private, cf. [5, 16, 20, 25]. Given the rising use of PDS in potentially malicious settings, there is an urgent need to better understand their performance in such settings, and to secure them against adversarial manipulation.

1.1 Our Contributions

In this paper, we ask (and answer) the question: *How can we provably protect PDS against attacks at low cost?* Given the vast range of PDS in the literature and in use, we focus our attention here on AMQ-PDS. These are amongst the most widely used PDS, with applications in dictionaries, databases and networking [10]. Bloom filters in particular are in common use, because of their implementation simplicity and easily understood performance guarantees; Cuckoo filters were more recently introduced [17] and offer superior performance to Bloom filters.

Syntax for AMQ-PDS. After establishing a syntax for AMQ-PDS, inspired by [12], we surface *consistency rules* satisfied by Bloom and insertion-only Cuckoo filters which we use to prove security theorems.

Simulation-based security. We then develop simulation-based security definitions for analysing the security of AMQ-PDS. Such an approach was recently used to study cardinality estimation [33], an important (but distinct) application domain for PDS. Inspired by [33], our approach introduces two worlds, a real world in which the adversary interacts with an AMQ-PDS instantiation through an API (allowing it to e.g. insert items and make membership queries) to produce some output, and an ideal world in which a simulator $\mathcal{S}$ produces the output. Our core security definitions say that the two outputs, one taken from each world, should be close.[1] The intuition is that whatever the adversary can learn from interacting with the AMQ-PDS in the real world can be simulated without access to the AMQ-PDS instantiation. The only leakage, then, is whatever input $\mathcal{S}$ receives. Security comes from carefully evaluating this input and/or restricting the simulator to behave in certain ways which imply it is effectively operating in a non-adversarial manner.

Our simulation-based approach appears conceptually complex, but it enables us to provide a fairly unified approach to both correctness (under adversarial input) and privacy of AMQ-PDS. It avoids the use of very specific winning conditions of the type used in [12], since security is defined purely in terms of closeness between the real and ideal world. For an introduction to the simulation-based approach, we recommend [27].

Adversarial correctness. We begin by establishing what degree of correctness it is reasonable to expect from an AMQ-PDS under non-adversarial inputs. Intuitively, we want to capture the behaviour of a given AMQ-PDS under "honest" or "non-adversarial" inputs. In principle, the distribution of queries that an honest user makes to a PDS will be application-dependent: a PDS deployed to store images will receive different inputs than one deployed to store IP addresses. However, we identify two properties of AMQ-PDS (satisfied by Bloom and insertion-only Cuckoo filters) that allow us to predict the state of an AMQ-PDS under honest inputs, in an application-independent manner. We use these to formalise the notion of a non-adversarially-influenced (NAI) state generator for an AMQ-PDS, and then define the NAI false positive probability. Essentially, this is the rate at which the AMQ-PDS incorrectly answers membership queries after having been populated with n randomly chosen elements (where n is a parameter of the definition), reflecting "average case" performance of the AMQ-PDS. This quantity can be computed or bounded from above for Bloom and Cuckoo filters, which will be sufficient for our purposes.

With this machinery in hand, we define adversarial correctness for AMQ-PDS in the following way: an AMQ-PDS is adversarially correct if there exists an ideal world simulator that provides a view of the AMQ-PDS to the adversary that is close to the view produced by an NAI generator. The definition implies that the adversary cannot tell the difference between the real world setting and the ideal world setting. Moreover, in the ideal world setting, the adversary effectively sees an AMQ-PDS operating under non-adversarial conditions. It follows from this chain of reasoning that, if our definition is satisfied, an adversary can only influence the false positive probability of the AMQ-PDS to a concretely bounded extent. Here, one should contrast the security definition with those more commonly seen in the simulation-based paradigm: we get security by restricting the *behaviour* of $\mathcal{S}$ rather than by limiting its *input*. An analogous formulation was used in [33] when dealing with adversarial correctness of cardinality estimators.

When it comes to proving application-independent results and using our adversarial correctness definition to study specific AMQ-PDS, we identify *function-decomposability* as a key property. Informally, this property says that the input to an AMQ-PDS is always first transformed using some function F before any further processing is applied. Bloom filters operate in this way (F represents the collection of hash functions used to map the input element to a vector of indices). Cuckoo filters do not, but we show how they can easily be modified to do so. Indeed, any existing AMQ-PDS can have its inputs "wrapped" in order to make it function-decomposable. Our main result shows that if an AMQ-PDS is function-decomposable, and F is a truly random function, then our notion of adversarial correctness is achievable. In practice, we instantiate F by a pseudorandom function (PRF), with the PRF key being held securely by the AMQ-PDS. This step introduces a PRF-to-random-function switching cost to our security bounds.[2]

Privacy. We consider two distinct but related security models for privacy of AMQ-PDS. The main question our models address is: what information can leak to an adversary about the elements already contained in an AMQ-PDS? Our first model recognises that such leakage may be inevitable if the adversary has access to a sufficiently rich API for a target AMQ-PDS (in particular, if it can make membership queries to the AMQ-PDS), but that it may be possible to prove that nothing more than the results of such queries may leak. This necessitates equipping the simulator with an oracle telling when a queried element is already contained in the AMQ-PDS. The second privacy model dispenses with this oracle, and only allows the simulator to learn *how many* elements have been stored in the AMQ-PDS. It captures the intuitive idea that

[1] Standard cryptographic notions here would mandate "indistinguishable" in place of "close". Our results do not reach this level for concrete AMQ-PDS like Bloom and insertion-only Cuckoo filters without blowing up the AMQ-PDS parameters, e.g. the required storage. Yet they are still useful when it comes to setting parameters in practice, for example, when limiting the false positive probability that an adversary can attain.

[2] Note that we are not operating in the Random Oracle Model here, since the adversary does not have (direct) oracle access to F; a good analogy is constructing a symmetric encryption mode using a random permutation π for the analysis, and then replacing π with a block cipher.

the only way an adversary should be able to break privacy is by guessing any of the previously stored elements. As one application, our security definition ensures that, provided the previously stored set has sufficiently high min-entropy, the adversary learns nothing about that set during its attack, even when equipped with a rich API for interacting with the AMQ-PDS, and even if it can compromise the AMQ-PDS' state. We show how these two models are related.

We also identify a property of AMQ-PDS that we call *permutation invariance (PI)*. This property enables us to quickly establish privacy according to our definitions. Informally, PI says that an adversary cannot tell if a random permutation has been applied to the elements before they are added to the AMQ-PDS, or not. We are able to show that PI follows from function-decomposability, so the latter property useful to establishing both adversarial correctness and privacy of an AMQ-PDS.

Analysis of Bloom and Cuckoo filters. Our final contribution is to use our security models to analyse two concrete AMQ-PDS, namely Bloom and insertion-only Cuckoo filters. Neither is secure according to our definitions in their original formulations, but we show how they can be made so, provably and at low cost, by replacing hash functions used in their constructions with keyed Pseudo-Random Functions (PRFs). Thus we invoke the use of cryptographic objects to achieve security. This is not completely cost-free of course, since now users must manage the needed cryptographic keys. However, PRFs are in general only a little more expensive to compute than hash functions (since a PRF on a given domain $\mathfrak{D}$ can always be constructed by first hashing on $\mathfrak{D}$ to obtain a fixed-length string and then applying a fixed-domain PRF, e.g. one based on a block cipher like AES, and fast, dedicated PRF constructions also exist [2]). Since our results do not require "indistinguishability" of the real and ideal worlds, one could potentially design faster and cheaper "weaker" PRFs apt for use in AMQ-PDS. Moreover, [30] have shown that adversarially correct Bloom filters *imply* one-way functions, in a weaker attack model than ours. Thus, using cryptographic approaches appears to be necessary for security of AMQ-PDS. We use our analysis to compute concrete bounds on how much security can be expected from PRF-based Bloom and insertion-only Cuckoo filters. These bounds can be used by practitioners who need to invoke AMQ-PDS in adversarial settings to choose appropriate parameters for their data structures.

1.2 Related Work

Clayton *et al.* [12] provided a provable-security treatment of PDS. They used a game-based formalism to analyse the correctness of Bloom filters, counting (Bloom) filters and count-min sketch data structures under adversarial inputs. Their approach required the introduction of a bespoke winning condition for the adversary and only addressed correctness. Our simulation-based approach dispenses with such a winning condition and allows us to consider both correctness and privacy in a single framework, at the cost of slightly worse bounds for their success condition of choice. We will consider their most powerful adversarial setting, where the adversary can access the internal state of the PDS and insert new items after an initial setup. We provide a more detailed comparison of our correctness results in Section 4.3.

Naor and Yogev [30, 31] analyse adversarial correctness (but not privacy) of Bloom filters in a game-based setting in which the capabilities of the adversary are strictly weaker than ours (the adversary can initialise the filter with a set of items and then make membership queries, but cannot access the internal state of the filter or make further insertions). They explore the relations between secure Bloom filters and one-way functions. They show that preprocessing the inputs to a Bloom filter using a PRP achieves adversarial correctness. This is similar to our function-decomposability result (but limited to Bloom filters and in a weaker attack model).

Earlier work on using Bloom filters in combination with cryptographic techniques to build searchable encryption (SE) schemes includes [3, 7, 22, 32, 35]. In particular, [22] cleverly combines Bloom filters with PRFs to build secure indexes, while [32] introduces a simulation-based security model for analysing a privacy-enhanced Bloom filter using blind signatures and oblivious PRFs (however, the model in [32] is weaker than ours, since it does not allow any new insertions after initialisation). SE schemes target outsourced storage of data with added functionality (like searchability); this involves complex two (and multi-) party cryptographic protocols. On the other hand, our work targets making secure versions of PDS that are widely deployed in real-world computing systems. A key difference is that the target in SE schemes is achieving privacy for users against a malicious server, whereas we focus on privacy (and adversarial correctness) of PDS against malicious users.

Paterson and Raynal [33] considered the HyperLogLog (HLL) cardinality estimator [18], introducing attacks and simulation-based definitions to study the correctness of HLL under adversarial inputs. Our approach is inspired by that of [33] but applies to a broad class of AMQ-PDS rather than to a specific cardinality estimator. In fact, the functionality of AMQ-PDS hinders the analysis of their behaviour in adversarial settings; the information revealed by membership queries can be leveraged to make adaptive insertions, resulting in more complicated proofs than in the HLL case.

Prior attack work has focused on Bloom filters [19], flooding of hash tables [15, 28], or key collision attacks on hash tables [11]. References [12, 19] provide partial summaries of prior work on the security of PDS, focused on Bloom filters. Work on privacy properties of Bloom filters can be found in [5, 20, 25].

A recent line of work [4, 24, 36] has studied broad classes of streaming algorithms under adversarial input. The models used are game-based and limit the adversary's capabilities (no corruption of the data structure is permitted). However, they are able to avoid introducing cryptographic assumptions, at the cost of producing schemes that are inefficient in practice. Our approach involves replacing hash functions in existing constructions with keyed PRFs, and so does involve cryptographic building blocks, but remains fully practical. Meanwhile, [23] considers adding robustness to streaming algorithms using differential privacy.

1.3 Paper Organisation

After preliminaries in § 2, § 3 develops our syntax for AMQ-PDS and formalises the NAI concept that we use to characterise correctness in the non-adversarial setting. Adversarial correctness is addressed in § 4 and privacy in § 5. In § 6 we concretely evaluate the security

our theorems provide when instantiating specific AMQ-PDS using pseudorandom functions.

2 PRELIMINARIES

Notation. Given an integer $m \in \mathbb{Z}_{\geq 1}$, we write $[m]$ to mean the set $\{1, 2, \ldots, m\}$. We consider all logarithms to be in base 2. If S is a set, we denote by $\mathcal{P}(S)$ the power set of S, and $\mathcal{P}_{\text{lists}}(S)$ the set of all lists with non-repeated elements from S. Given a statement S, we denote by $[S]$ the function that returns $\top$ if S is true and $\bot$ otherwise. Given two sets $\mathfrak{D}$ and $\mathfrak{R}$, we define $\mathsf{Funcs}[\mathfrak{D}, \mathfrak{R}]$ to be the set of functions from $\mathfrak{D}$ to $\mathfrak{R}$. We write $F \leftarrow\!\$\ \mathsf{Funcs}[\mathfrak{D}, \mathfrak{R}]$ to mean that F is a random function $\mathfrak{D} \xrightarrow{F} \mathfrak{R}$. Given a set $\mathfrak{D}$ we define $\mathsf{Perms}[\mathfrak{D}]$ to be the set of permutations over $\mathfrak{D}$. We write $\pi \leftarrow\!\$\ \mathsf{Perms}[\mathfrak{D}]$ to mean that π is a random permutation over $\mathfrak{D}$. Given a set S, we denote the identity function over S as $\mathrm{Id}_S \colon S \to S$. If D is a probability distribution, we write $x \leftarrow\!\$\ D$ to mean that x is sampled according to D. We denote the uniform distribution over a finite set S as $U(S)$. Given a set S (resp. a list L), we denote by $|S|$ (resp. $|L|$) the number of elements in S (resp. L). To indicate a fixed-length list of length s initialised empty, we write $a \leftarrow \bot^s$. We let $\mathrm{load}(a)$ be the number of set entries of a. To insert an entry x into the first unused slot in a we write $a' \leftarrow a \diamond x$ such that $a' = x \bot \ldots \bot$ with $s-1$ trailing $\bot$s and $\mathrm{load}(a') = 1$. A further insertion $a'' \leftarrow a' \diamond y$ results in $a'' = x\, y \bot \ldots \bot$ with $\mathrm{load}(a'') = 2$, and so on. We refer to the i-th entry in a list a as $a[i]$. In algorithms, we assume that all key-value stores are initialised with value $\bot$ at every index, using the convention that $\bot < n$, $\forall n \in \mathbb{R}$, and we denote it as $\{\}$. Given a key-value store a, we refer to the value of the entry with key k as $a[k]$, and we refer to the set of all inserted values as $\mathrm{vals}(a)$. We write variable assignments using $\leftarrow$, unless the value is output by a randomised algorithm, for which we use $\leftarrow\!\$$.

For any randomised algorithm alg, we may denote the coins that alg can use as an extra argument $r \in \mathcal{R}$ where $\mathcal{R}$ is the set of possible coins, and write $\mathit{output} \leftarrow \mathrm{alg}(\mathrm{input}_1, \mathrm{input}_2, \ldots, \mathrm{input}_\ell; r)$. We may also suppress coins whenever it is notationally convenient to do so. If an algorithm is deterministic, we allow setting r to $\bot$. We remark that the output of a randomised algorithm can be seen as a random variable over the output space of the algorithm. Unless otherwise specified, we will consider random coins to be sampled uniformly from $\mathcal{R}$, independently from all other inputs and/or state. We may refer to such r as "freshly sampled". We write $\mathrm{alg}^{f_1,\ldots f_n}$ whenever alg is given oracle access to functions $f_1, \ldots, f_n$.

In this work we will consider AMQ-PDS that can store elements from finite domains $\mathfrak{D}$ by letting $\mathfrak{D} = \cup_{\ell=0}^{L} \{0,1\}^\ell$ for some large but finite value of L, say $L = 2^{64}$. This allows us to give a natural definition of the setting where an AMQ-PDS is not influenced by an adversary in Def. 3.3. Our constructions make use of pseudorandom functions, which we model as truly random functions to which the AMQ-PDS has oracle access. While a priori multiple random functions may be used by a particular AMQ-PDS, our results are stated in terms of a single function $F \colon \mathfrak{D} \to \mathfrak{R}$, with $\mathfrak{D}$ being the finite set described above, and $\mathfrak{R}$ depending on the AMQ-PDS algorithms and public parameters. We will see that in practice this will not be a limitation, even in the case of insertion-only Cuckoo filters, originally described as using two hash functions.

Definition 2.1. We say a pseudorandom function family $R \colon \mathcal{K} \times \mathfrak{D} \to \mathfrak{R}$ is (q, t, ε)-secure if any adversary $\mathcal{B}$ running in time at most t and making at most q queries to either R_K where $K \leftarrow\!\$\ \mathcal{K}$ or a random function $F \leftarrow\!\$\ \mathrm{Func}[\mathfrak{D}, \mathfrak{R}]$, has distinguishing advantage:

$$\mathrm{Adv}_R^{PRF}(\mathcal{B}) := |\Pr[\mathcal{B}^{R_K(\cdot)} = 1] - \Pr[\mathcal{B}^{F(\cdot)} = 1]| \leq \varepsilon.$$

We say $\mathcal{B}$ is a (q, t)-PRF adversary.

Probabilistic Data Structures (PDS). Different PDS have been proposed to efficiently provide approximate responses to various kinds of queries. For example, Bloom filters [6] provide answers to approximate membership queries (AMQs); that is, they answer queries of the form $x \stackrel{?}{\in} S$ for some finite set S. Count-min sketches [14] provide approximate frequency estimates for events in a data stream, and HyperLogLog [18] computes the approximate number of distinct elements in a data stream. For each problem, multiple PDS designs have been proposed, with a tradeoff between their performance and the features they offer. For example, while Bloom filters provide AMQ answers for a set $S \subset \mathfrak{D}$ under the assumption that items can only be inserted into S but not removed, Cuckoo filters [17] also allow deletion of items, but at the cost of introducing false negative errors.

The specific design of a PDS implies certain *consistency rules* that the data structure will satisfy. For example, Bloom filters provide the guarantee of having no false negatives, which means they will never answer "false" to a membership query on x when $x \in S$. Consistency rules will be important when we prove security properties of PDS; if they are not respected, an adversary in a security game may be able to observe inconsistent behaviour and tell they are interacting with a simulation of the PDS rather than the real PDS.

In this paper, we will study PDS that rely on hash functions to provide good expected behaviour on non-adversarial inputs. In our proofs, we will replace such hash functions with random functions, and eventually replace these with keyed pseudorandom functions. We refer to the latter as *keyed* PDS. We focus on insertion-only AMQ-PDS, i.e. those that support insertions and membership queries, but not deletions.

3 INSERTION-ONLY AMQ-PDS

We proceed to define the syntax of an insertion-only AMQ-PDS. We remark that we only consider AMQ-PDS with deterministic membership checks that do not modify the state of the AMQ-PDS, since these include the two most popular AMQ-PDS, namely Bloom and (insertion-only) Cuckoo filters.

We denote public parameters for an AMQ-PDS by pp. We denote the state of a PDS Π as $\sigma \in \Sigma$, where Σ denotes the space of possible *states* of Π. The set of elements that can be inserted into the AMQ-PDS is denoted by $\mathfrak{D}$, unless stated otherwise. We consider a syntax consisting of three algorithms:

- The setup algorithm $\sigma \leftarrow \mathrm{setup}(pp; r)$ sets up the initial state of an empty PDS with public parameters pp; it will always be called first to initialise the AMQ-PDS.
- The insertion algorithm $(b, \sigma') \leftarrow \mathrm{up}(x, \sigma; r)$, given an element $x \in \mathfrak{D}$, attempts to insert it into the AMQ-PDS, and returns a bit $b \in \{\bot, \top\}$ representing whether the insertion was successful ($b = \top$) or not ($b = \bot$), and the state σ' of the AMQ-PDS after the insertion.

- The membership querying algorithm $b \leftarrow \mathsf{qry}(x, \sigma)$, given an element $x \in \mathfrak{D}$, returns a bit $b \in \{\bot, \top\}$ (approximately) answering whether x was previously inserted ($b = \top$) or not ($b = \bot$) into the AMQ-PDS.

We note that qry does not change the state of the AMQ-PDS, and hence does not output a new σ' value.

In the case of Bloom and Cuckoo filters, qry calls may return a false positive result with a certain probability. That is, we may have $\top \leftarrow \mathsf{qry}(x, \sigma)$ even though no call $\mathsf{up}(x, \sigma'; r)$ was made post setup and prior to the membership query. We refer to the probability $\Pr[\top \leftarrow \mathsf{qry}(x, \sigma) \mid x \text{ was not inserted into } \Pi]$ as the *false positive probability* of an AMQ-PDS Π. This probability depends on σ, which could be generated as the result of a sequence of adversarially chosen insertions, or which could just arise through insertions made by honest users.

AMQ-PDS in the honest setting. To argue about how the state of an AMQ-PDS under adversarial queries may deviate from expected in a "honest" or "non-adversarial" setting, we first define what we expect to see in an honest setting. In general, non-adversarial input distributions are application-dependent. For example, honest users may sample inputs from distributions with different entropy in different applications. Similarly, different applications may imply that domain elements may be likely to be inserted multiple times, or just once. This suggests that proving results for arbitrary AMQ-PDS may require defining the distributions produced by non-adversarial actors, in order to quantify the expected performance of the data structure under honest inputs.

We overcome this issue by noticing two properties that an AMQ-PDS can satisfy and that suffice to argue about their performance under non-adversarial inputs in an application-independent manner: function-decomposability and reinsertion invariance.

Definition 3.1 (Function-decomposability). Let Π be an insertion-only AMQ-PDS and let $F \leftarrow\!\!\$\ \mathsf{Funcs}[\mathfrak{D}, \mathfrak{R}]$ with $\mathfrak{R} \subset \mathfrak{D}$ be a random function to which Π has oracle access. Let $\mathrm{Id}_{\mathfrak{R}}$ be the identity function over $\mathfrak{R}$. We say that Π is F-decomposable if we can write

$$\mathsf{up}^{F}(x, \sigma; r) = \mathsf{up}^{\mathrm{Id}_{\mathfrak{R}}}(F(x), \sigma; r) \quad \forall x \in \mathfrak{D},\ \sigma \in \Sigma,\ r \in \mathcal{R},$$
$$\mathsf{qry}^{F}(x, \sigma) = \mathsf{qry}^{\mathrm{Id}_{\mathfrak{R}}}(F(x), \sigma) \quad \forall x \in \mathfrak{D},\ \sigma \in \Sigma,$$

where $\mathsf{up}^{\mathrm{Id}_{\mathfrak{R}}}$ and $\mathsf{qry}^{\mathrm{Id}_{\mathfrak{R}}}$ cannot internally evaluate F due to not having oracle access to it and F being truly random.

Function-decomposability also applies to AMQ-PDS with oracle access to multiple functions. For example, if Π had oracle access to t functions $F_1, \ldots, F_t$ and was F_1-decomposable, we would write

$$\mathsf{up}^{F_1,\ldots,F_t}(x, \sigma; r) = \mathsf{up}^{\mathrm{Id}_{\mathfrak{R}},F_2,\ldots,F_t}(F_1(x), \sigma; r)\ \forall x, \sigma, r,$$
$$\mathsf{qry}^{F_1,\ldots,F_t}(x, \sigma) = \mathsf{qry}^{\mathrm{Id}_{\mathfrak{R}},F_2,\ldots,F_t}(F_1(x), \sigma)\ \forall x, \sigma.$$

Function-decomposability has the effect of "erasing" any structure on the input domain $\mathfrak{D}$. Essentially, function-decomposable AMQ-PDS replace any input element $x \in \mathfrak{D}$ with a fixed element $y \in \mathfrak{R}$ sampled uniformly at random, and proceed to do any further processing on y. This allows us to disregard the input distribution on $\mathfrak{D}$ and instead think of input elements sampled uniformly at random from $\mathfrak{R}$.

In § 4 and § 5, we will prove theoretical guarantees for AMQ-PDS instantiated using pseudorandom functions (PRFs). After a switch from PRFs to truly random functions, we will be able to assume function-decomposability.

n-NAI-gen$^{F_1,\ldots,F_t}(pp)$

1 $\sigma^{(0)} \leftarrow\!\!\$\ \mathsf{setup}(pp)$
2 $[x_1, \ldots, x_n] \leftarrow\!\!\$\ \{S \in \mathcal{P}_{\mathrm{lists}}(\mathfrak{D}) \mid |S| = n\}$
3 **for** $j = 1, \ldots, n$: $\quad (b, \sigma^{(j)}) \leftarrow\!\!\$\ \mathsf{up}^{F_1,\ldots,F_t}(x_j, \sigma^{(j-1)})$
4 **return** $\sigma^{(n)}$

Figure 1: Algorithm returning non-adversarially-influenced (NAI) state.

Definition 3.2 (Reinsertion invariance). Consider an AMQ-PDS Π. We say Π is *reinsertion invariant* if for all $x \in \mathfrak{D}$, $\sigma \in \Sigma$ such that $\top \leftarrow \mathsf{qry}(x, \sigma)$, we have $(b, \sigma') \leftarrow \mathsf{up}(x, \sigma; r) \implies \sigma = \sigma'\ \forall r \in \mathcal{R}$. Informally, if x appears to have been inserted, then further insertions of x will not cause the state σ of Π to change.

This property is shared by Bloom and insertion-only Cuckoo filters, and it appears natural since insertion-only AMQ-PDS aim to represent sets and not multisets; hence repeated insertions need not change the state.

The state of a function-decomposable, reinsertion invariant AMQ-PDS is not affected by any structure on the input elements sampled from $\mathfrak{D}$, nor by elements being reinserted more than once. Rather, it only depends on the number of distinct elements in the data structure. This allows us to define the notion of an (n, ε)-non-adversarially-influenced state as follows.

Definition 3.3 ((n, ε)-NAI). Let $\varepsilon > 0$, and let n be a non-negative integer. Let Π be an AMQ-PDS with public parameters pp and state space Σ, such that its up algorithm makes use of oracle access to functions $F_1, \ldots, F_t$. Let alg be a randomised algorithm outputting values in Σ. Let σ and $\sigma^{(n)}$ be random variables representing respectively the outputs of alg and of the randomised algorithm n-NAI-gen described in Fig. 1. We say that alg outputs an (n, ε)-non-adversarially-influenced state (denoted by (n, ε)-NAI) if σ is ε-statistically close to $\sigma^{(n)}$.

In Def. 3.3, the n-NAI-gen algorithm imitates the behaviour of an honest user inserting distinct elements into the PDS.[3] The distribution output by n-NAI-gen then becomes the benchmark for how close to honestly-generated (or non-adversarially-influenced, NAI) the state of an AMQ-PDS is. Now we are ready to introduce the NAI false positive probability for an AMQ-PDS.

Definition 3.4 (NAI false positive probability). Let Π be a function-decomposable reinsertion invariant AMQ-PDS with public parameters pp, using functions $F_1, \ldots, F_t$ sampled respectively from distributions $D_{F_1}, \ldots, D_{F_t}$ to instantiate its functionality. Let n be a non-negative integer. Define the *NAI false positive probability after n distinct insertions* as

$$P_{\Pi,pp}(FP \mid n) := \Pr\left[\begin{array}{l} F_1 \leftarrow\!\!\$\ D_{F_1}, \ldots, F_t \leftarrow\!\!\$\ D_{F_t}, \\ \sigma \leftarrow\!\!\$\ n\text{-NAI-gen}(pp), \\ x \leftarrow\!\!\$\ \mathfrak{D} \setminus V \end{array} : \top \leftarrow \mathsf{qry}^{F_1,\ldots,F_t}(x, \sigma)\right],$$

[3] Note that in the case of Cuckoo filters, not all insertions may succeed; see discussion later in this section.

$\mathsf{setup}(pp)$	$\mathsf{up}^F(x,\sigma)$	$\mathsf{qry}^F(x,\sigma)$
1 $m,\ k \leftarrow pp;\ \sigma \leftarrow 0^m$	1 $\sigma' \leftarrow \sigma \vee B_{m,k}(F(x))$	1 $b \leftarrow B_{m,k}(F(x))$
2 **return** σ	2 **return** $\top, \sigma'$	2 **return** $[b = \sigma \wedge b]$

Figure 2: AMQ-PDS syntax instantiation for the Bloom filter.

where V is the list $[x_1, \ldots, x_n]$ sampled on line 2 of n-NAI-gen(pp).

Definition 3.4 captures the probability that a non-adversarial user experiences a false positive membership query result after inserting n distinct elements into Π.

Bloom filters. The most popular AMQ-PDS is the Bloom filter [6]. Informally, a Bloom filter consists of a bitstring σ of length m initially set to 0^m, and a family of k independent hash functions $H_i : \{0,1\}^* \to [m]$, for $i \in [k]$. An element x is inserted into the filter by setting bit $H_i(x)$ of σ to 1, for every $i \in [k]$. A membership query on x is carried out by checking if the k bits $H_i(x), i \in [k]$ are set to 1 in σ. Bloom filters have no false negatives, i.e. if x has been inserted then a membership query on x always returns $\top$. However, Bloom filters can have false positives, i.e. a membership query on x can return $\top$ even when x has not been inserted, due to the potential for collisions in the hash functions H_i.

In this work, we bundle the k hash functions H_i into a single function $F\colon \mathfrak{D} \to \mathfrak{R} = [m]^k$. We will later instantiate F with a pseudorandom function, rather than a fixed hash function, to achieve our security notions. This bundling is convenient for making our formal description of a Bloom filter (that follows) fit with our general AMQ-PDS syntax. We now formally define Bloom filters.

Definition 3.5. Let $B_{m,k} : [m]^k \to \{0,1\}^m$ be the map that on input $\vec{x} \in [m]^k$ returns an m-bit "bitmap" where all bits are zero except those at the indices $x_i, i \in [k]$.

Definition 3.6. Let $m,\ k$ be positive integers. We define an (m,k)-Bloom filter to be the AMQ-PDS with algorithms defined in Fig. 2, with $pp = (m,k)$, and $F\colon \mathfrak{D} \to \mathfrak{R} \equiv [m]^k$.

We now recall from the literature an estimate for and an upper bound on the NAI false positive probability for Bloom filters.

Lemma 3.7. *([6],[21, Theorem 4.3]) Let Π be an (m,k)-Bloom filter using a random function $F\colon \mathfrak{D} \to [m]^k$. Define $\overline{P_{\Pi,pp}}(FP \mid n) := \left(1 - e^{-\frac{(n+0.5)k}{m-1}}\right)^k$. Then for any n, 1) $P_{\Pi,pp}(FP \mid n) \xrightarrow{m\to\infty} (1 - e^{-nk/m})^k$, and 2) $\overline{P_{\Pi,pp}}(FP \mid n) \geq P_{\Pi,pp}(FP \mid n)$.*

Insertion-only Cuckoo filters. Proposed as an improvement over Bloom filters, Cuckoo filters [17] allow deletion of elements, at the cost of potentially introducing false negatives if a user attempts to delete an element that was not previously inserted. In [17, §3.1], the authors also consider the case of *insertion-only Cuckoo filters*, a variant that does not support deletions. In their original description, insertion-only Cuckoo filters use two hash functions $H_I\colon \mathfrak{D} \to \{0,1\}^{\lambda_I}$ and $H_T\colon \mathfrak{D} \to \{0,1\}^{\lambda_T}$. Let σ consist of a collection $(\sigma_i)_i$ of $m = 2^{\lambda_I}$ fixed-length lists, or "buckets", indexed by $i \in [m]$ and each containing s slots, together with a stash σ_{evic} containing one more slot.[4] A detailed description of the internals of Cuckoo filters can be found in the full version. In contrast to Bloom filters, two hash functions H_T and H_I are used at different points in the up and qry algorithms of Cuckoo filters. Hence, instead of bundling them into a single function F, we will give up and qry access to both functions. In § 4.2, we will discuss different approaches to replacing these with PRFs in order to achieve our security notions.

Definition 3.8. Let $pp = (s, \lambda_I, \lambda_T, num)$ be a tuple of positive integers. We define an $(s, \lambda_I, \lambda_T, num)$-Cuckoo filter to be the AMQ-PDS with algorithms defined in the full version, making use of hash functions $H_T\colon \mathfrak{D} \to \{0,1\}^{\lambda_T}$ and $H_I\colon \mathfrak{D} \to \{0,1\}^{\lambda_I}$.

In the original definition of [17], the false positive probability of a Cuckoo filter with all its buckets full is computed assuming H_T is a random function, and is given by $1 - (1 - 2^{-\lambda_T})^{2s+1}$. In [17], this value is shown to be an upper bound on the false positive probability of a Cuckoo filter containing $n \leq m \cdot s$ elements.

Lemma 3.9. *([17]) Let Π be a $(s, \lambda_I, \lambda_T, num)$-Cuckoo filter and let $H_T\colon \mathfrak{D} \to \{0,1\}^{\lambda_T}$ be a random function. For any non-negative integer n, $\overline{P_{\Pi,pp}}(FP \mid n) := 1 - \left(1 - 2^{-\lambda_T}\right)^{2s+1} \geq P_{\Pi,pp}(FP \mid n)$.*

While Bloom and Cuckoo filters instantiate the insertion-only AMQ-PDS syntax described above, they also satisfy some additional properties that we refer to as "consistency rules", captured below.

Definition 3.10 (Insertion-only AMQ-PDS consistency rules). Consider an AMQ-PDS Π. We say Π has:

- *Element permanence* if for all $x \in \mathfrak{D},\ \sigma \in \Sigma$ such that $\top \leftarrow \mathsf{qry}(x,\sigma)$, and for any sequence of insertions resulting in a later state $\sigma',\quad b \leftarrow \mathsf{qry}(x,\sigma') \implies b = \top$.
- *Permanent disabling* if given $\sigma \in \Sigma$ such that there exists $x \in \mathfrak{D}, r \in \mathcal{R}$ where $(b, \overline{\sigma}) \leftarrow \mathsf{up}(x,\sigma;r)$ and $b = \bot$, then $\overline{\sigma} = \sigma$ and for any $x' \in \mathfrak{D}, r' \in \mathcal{R},\quad (b', \sigma') \leftarrow \mathsf{up}(x',\sigma;r') \Rightarrow b' = \bot$ and $\sigma' = \sigma$.
- *Non-decreasing membership probability* if for all $\sigma \in \Sigma, x, y \in \mathfrak{D}, r \in \mathcal{R},\quad (b,\sigma') \leftarrow \mathsf{up}(x,\sigma;r) \Rightarrow \Pr[\top \leftarrow \mathsf{qry}(y,\sigma)] \leq \Pr[\top \leftarrow \mathsf{qry}(y,\sigma')]$.

4 ADVERSARIAL CORRECTNESS

In this section, we develop simulation-based security definitions with which we then analyse the adversarial correctness of AMQ-PDS. We derive bounds on the correctness of insertion-only AMQ-PDS that are function-decomposable, reinsertion invariant, and obey the consistency rules in Def. 3.10. Finally, we apply our results to provide correctness guarantees for PRF-instantiated Bloom filters and a straightforward variant of insertion-only Cuckoo filters. Amongst other things, these guarantees limit an adversary's ability to carry out pollution attacks [19] and target-set coverage attacks [12] on Bloom filters. Both attacks involve an adversary manipulating the false positive probability of a Bloom filter, and our analysis shows that this is not possible (up to some security bounds that we make concrete in § 6).

[4] We note that while no restrictions on m are mentioned explicitly in [17], it was recently pointed out [8] that m should be a power of two in order to avoid the potential introduction of false negative results.

Settings. Our model (see Fig. 3) considers an adversary $\mathcal{A}$ interacting with an AMQ-PDS Π in two stages. In the first stage, the data structure is initialised empty, and $\mathcal{A}$ provides a finite set of elements to insert into it through the **Rep** oracle, which can be called only once. In the second stage, the adversary is given access to three other oracles: **Qry**, responding to queries of the form "has element x been inserted into Π?", **Up**, inserting an element provided by $\mathcal{A}$ into Π, and **Reveal**, returning Π's current state.

While both the **Rep** and **Up** oracles allow insertion of elements, defining these as separate oracles allows us to treat initialisation and subsequent insertions/queries as distinct stages. Then, by disabling access to the **Up** oracle, we can define an *immutable* setting in which no insertions are allowed after initialisation, as in [12]. Further, this separation of **Rep** and **Up** oracles will come in useful in our treatment of privacy in § 5.

4.1 Notions of Correctness

Our analysis of the adversarial correctness of AMQ-PDS uses a simulation-based approach. We start with a high-level explanation of our approach in order to provide some intuition.

In our security framework, the adversary $\mathcal{A}$ plays in either the real or ideal world. In the real world, it interacts with a keyed AMQ-PDS Π, where it has access to oracles that allow it to insert elements into the data structure, as well as to make membership queries for elements of its choice. In the ideal world, it interacts with a simulator $\mathcal{S}$, constructed so as to provide an NAI view of Π to $\mathcal{A}$. At the end of its execution, $\mathcal{A}$ produces some output, which is given to a distinguisher $\mathcal{D}$. $\mathcal{A}$'s output is arbitrary – for example, it could be the state of the AMQ-PDS obtained by making an appropriate oracle query. $\mathcal{D}$'s task is to compute which world $\mathcal{A}$ was operating in, based on its output. By bounding $\mathcal{D}$'s ability to distinguish between the ideal and real worlds, we can quantify how much more harm $\mathcal{A}$ can do in the real world (where it can make adaptive insertions and membership queries) compared to the ideal world (where everything is handled by $\mathcal{S}$ in an NAI manner).

We begin by defining the Real-or-Ideal game in Fig. 3. We will use *Real* and *Ideal* to denote the real ($d = 0$) and ideal ($d = 1$) versions of Real-or-Ideal, respectively. The game's output is the bit d' generated by a distinguisher $\mathcal{D}$ operating on $\mathcal{A}$'s output (which is an arbitrary string whose length is incorporated into the running time of $\mathcal{A}$). Throughout the paper, if an oracle **O** is not directly specified, we assume it is defined as in Fig. 3.

Definition 4.1. Let Π be an insertion-only AMQ-PDS, with public parameters pp, and let R_K be a keyed function family. We say Π is $(q_u, q_t, q_v, t_a, t_s, t_d, \varepsilon)$-adversarially correct if, for all adversaries $\mathcal{A}$ running in time at most t_a and making at most a single query to **Rep** and q_u, q_t, q_v queries to oracles **Up**, **Qry**, **Reveal** respectively in the Real-or-Ideal game (Fig. 3) with a simulator $\mathcal{S}$ *that provides an NAI view of* Π *to* $\mathcal{A}$ and runs in time at most t_s, and for all distinguishers $\mathcal{D}$ running in time at most t_d, we have:

$$\mathrm{Adv}^{RoI}_{\Pi,\mathcal{A},\mathcal{S}}(\mathcal{D}) := \left| \Pr[\, Real(\mathcal{A},\mathcal{D}){=}1 \,] - \Pr[\, Ideal(\mathcal{A},\mathcal{D},\mathcal{S}){=}1 \,] \right| \leq \varepsilon.$$

Remark 1. *While we explicitly only cover the case where the adversary calls the* ***Rep*** *oracle once, a hybrid argument could be used to derive a bound for the case with $q_r > 1$* ***Rep*** *queries, as long as the function F is resampled between* ***Rep*** *calls. This would come at the cost of introducing a factor q_r to our bounds.*

Real-or-Ideal$(\mathcal{A}, \mathcal{S}, \mathcal{D}, pp)$

1 $d \leftarrow\$ \{0,1\}$
2 **if** $d = 0$ // Real
3 $K \leftarrow\$ \mathcal{K}; F \leftarrow R_K$
4 init $\leftarrow \perp$
5 $\sigma \leftarrow \mathrm{setup}(pp)$
6 $out \leftarrow\$ \mathcal{A}^{\mathbf{Rep},\mathbf{Up},\mathbf{Qry},\mathbf{Reveal}}$
7 **else** // Ideal
8 $out \leftarrow\$ \mathcal{S}(\mathcal{A}, pp)$
9 **return** $d' \leftarrow\$ \mathcal{D}(out)$

Oracle **Rep**(V)

1 **if** init $= \top$: **return** $\perp$
2 init $\leftarrow \top$
3 **for** $x \in V$
4 $(b, \sigma) \leftarrow\$ \mathrm{up}^F(x, \sigma)$
5 **return** $\top$

Oracle **Up**(x)

1 **if** init $= \perp$: **return** $\perp$
2 $(b, \sigma) \leftarrow\$ \mathrm{up}^F(x, \sigma)$
3 **return** b

Oracle **Qry**(x)

1 **if** init $= \perp$
2 **return** $\perp$
3 **return** $\mathrm{qry}^F(x, \sigma)$

Oracle **Reveal**$()$

1 **return** σ

Figure 3: Correctness game for AMQ-PDS Π.

Remark 2. *We explain why our definition captures adversarial correctness. Consider an arbitrary adversary $\mathcal{A}$ that in the course of its execution makes* ***Up***$(\cdot)$ *queries on adversarially selected inputs $x_1, \ldots, x_n$. These are (potentially) interspersed with other types of query permitted to $\mathcal{A}$. Consider an extension of $\mathcal{A}$, named $\mathcal{A}^*$, that behaves exactly as $\mathcal{A}$ does, but which makes a final membership query* ***Qry***(x) *with $x \leftarrow\$ \mathfrak{D} \setminus \{x_1, \ldots, x_n\}$. Suppose the output of $\mathcal{A}^*$ is the result of that final query (a binary value), and $\mathcal{D}$'s output is identical to that of $\mathcal{A}^*$. Then it is easy to see that* $\Pr[\, Real(\mathcal{A}^*, \mathcal{D}) \,]$ *is exactly the* adversarial *false positive probability of* Π *produced by $\mathcal{A}$, while* $\Pr[\, Ideal(\mathcal{A}^*, \mathcal{D}, \mathcal{S}) \,]$ *is the NAI false positive probability. The definition, if satisfied, then says that these two probabilities must be within ε of each other. Even if ε cannot be shown to be very small for some specific AMQ-PDS, we may still obtain a useful result about adversarial false positive probability in practice. The above argument involves an arbitrary $\mathcal{A}$ and a specific $\mathcal{D}$. The reader may imagine that other choices of $(\mathcal{A}, \mathcal{D})$ may capture additional correctness properties. See § 4.3 for further discussion.*

The details of how the simulator is constructed (and how to bound the distinguishing advantage) depend on the data structure under consideration. Recall that we only consider AMQ-PDS that support insertions and membership queries, but not deletions. In Fig. 4, we give a simulator $\mathcal{S}$ that replicates the behaviour of AMQ-PDS that satisfy the consistency rules from Def. 3.10, are function-decomposable (see Def. 3.1) and reinsertion invariant (see Def. 3.2). By inspection, the runtime of $\mathcal{S}$ is not significantly higher than that of the underlying AMQ-PDS.

We now proceed to state and prove our correctness theorem.

Theorem 4.2. *Let q_u, q_t, q_v be non-negative integers, and let t_a, $t_d > 0$. Let $F\colon \mathfrak{D} \to \mathfrak{R}$. Let Π be an insertion-only AMQ-PDS with public parameters pp and oracle access to F, such that Π satisfies the consistency rules from Def. 3.10, F-decomposability (Def. 3.1) and reinsertion invariance (Def. 3.2). Let n be the number of elements provided by $\mathcal{A}$ for initial insertion into Π by a query call to* ***Rep***.[5] *Let α (resp. β) be the number of calls to F required to insert (resp. query) an element in Π using its up (resp. qry) algorithm.*

[5] Note there is no guarantee that all n elements are successfully inserted.

Simulator $\mathcal{S}(\mathcal{A}, pp)$

1 $F \leftarrow\$ \text{Funcs}[\mathfrak{D}, \mathfrak{R}]$
2 init $\leftarrow \bot$; UpIsEnabled $\leftarrow \top$
3 $\sigma \leftarrow \text{setup}(pp)$
4 inserted, FPlist, CALQ $\leftarrow \{\}, \{\}, \{\}$
5 $i \leftarrow 0$ // Qry counter
6 $ctr \leftarrow 0$ // Distinct insertions
7 $out \leftarrow\$ \mathcal{A}^{\textbf{RepSim, UpSim, QrySim, RevealSim}}$
8 **return** out

Oracle **RepSim**(V)

1 **if** init $= \top$: **return** $\bot$
2 init $\leftarrow \top$
3 **for** $x \in V$: $b \leftarrow$ **UpSim**(x)
4 **return** $\top$

Oracle **QrySim**(x)

1 **if** init $= \bot$: **return** $\bot$
2 $i \leftarrow i + 1$
3 // Element was inserted or determined a false positive
4 **if** inserted$[x] = \top$ or FPlist$[x] = \top$
5 **return** $\top$
6 // Element was not inserted and not false positive
7 **if** CALQ$[x] = ctr$ // If no changes since last query of x
8 **return** $\bot$
9 // Response needs to be (re)computed
10 CALQ$[x] \leftarrow ctr$
11 $a_i^{Ideal} \leftarrow\$ \text{qry}^{\text{Id}_{\mathfrak{R}}}(Y \leftarrow\$ \mathfrak{R}, \sigma)$
12 $a_i^{G^*} \leftarrow \text{qry}^F(x, \sigma)$
13 $a \leftarrow a_i^{Ideal}$ $\boxed{a \leftarrow a_i^{G^*}}$ // Ideal $\boxed{G^*}$
14 **if** $a = \top$: FPlist$[x] \leftarrow \top$
15 **return** a

Oracle **UpSim**(x)

1 **if** init $= \bot$: **return** $\bot$
2 **if** inserted$[x] = \bot$
3 $(b, \sigma) \leftarrow\$ \text{up}^F(x, \sigma)$
4 UpIsEnabled $\leftarrow b$
5 **if** $b = \top$
6 inserted$[x] \leftarrow \top$
7 $ctr \leftarrow ctr + 1$
8 **return** b
9 **else**
10 **return** UpIsEnabled

Oracle **RevealSim**$()$

1 **return** σ

Figure 4: Simulator $\mathcal{S}$ used in Theorem 4.2. Lines 12-13 corresponding to intermediate game G^* are used in our proof.

If $R_K : \mathfrak{D} \to \mathfrak{R}$ is an $(\alpha(n + q_u) + \beta q_t, t_a + t_d, \varepsilon)$-secure pseudorandom function with key $K \leftarrow\$ \mathcal{K}$, then Π is $(q_u, q_t, q_v, t_a, t_s, t_d, \varepsilon')$-adversarially correct with respect to the simulator in Fig. 4, where $\varepsilon' = \varepsilon + 2q_t \cdot P_{\Pi,pp}(FP \mid n + q_u)$ and $t_s \approx t_a$.

SKETCH. We start by defining an intermediate game G that replaces the PRF in *Real* with a random function. We then bound the closeness of *Real* and G in terms of the PRF advantage ε. To bound the distance between G and *Ideal*, we construct a game G^* (Fig. 4) that looks identical to G, and show that G^* and *Ideal* are equal up until the "bad" event that $a_i^{Ideal} \neq a_i^{G^*}$ for some $i \in [q_t]$. Then, we show that our simulator constructs an NAI view of Π in *Ideal*. Finally, we upper bound the probability of the bad event to obtain our result. The full proof is given in the full version. □

While Theorem 4.2 only refers to a single oracle function F for notational simplicity, the same result holds also for AMQ-PDS using t oracle functions $F_1, \ldots F_t$ and being F_1-decomposable. This requires adding sampling of the functions $F_1, \ldots F_t$ from distributions $D_{F_1}, \ldots D_{F_t}$, given by the specification of the AMQ-PDS, at the beginning of Real-or-Ideal (Fig. 3), and either allowing oracle access to $F_2, \ldots, F_t$ to $\mathcal{S}$ or sampling them also at the beginning of the simulator (Fig. 4). Then one would replace all calls to $\text{up}^F, \text{qry}^F, \text{qry}^{\text{Id}_{\mathfrak{R}}}$ with $\text{up}^{F_1,\ldots F_t}, \text{qry}^{F_1,\ldots F_t}, \text{qry}^{\text{Id}_{\mathfrak{R}},F_2,\ldots F_t}$. We stress that the proof would still only incur into the PRF-switching cost of F_1, since only F_1-decomposability is used. However, every $D_{F_1}, \ldots D_{F_t}$ would appear in the definition of $P_{\Pi,pp}(FP \mid n + q_u)$ and as such may directly influence the NAI false positive probability.

4.2 Guarantees for Bloom and Cuckoo filters

Our goal is to give bounds on the adversarial correctness of Bloom and Cuckoo filters. We will prove that Bloom and a straightforward variant of insertion-only Cuckoo filters satisfy the consistency rules from Def. 3.10, function-decomposability and reinsertion invariance. This in turn allows us to use Theorem 4.2 to provide concrete correctness guarantees.

Bloom filters. We start by proving the function-decomposability of Bloom filters.

LEMMA 4.3. *Bloom filters with oracle access to a random function F are F-decomposable, reinsertion invariant, and satisfy the insertion-only AMQ-PDS consistency rules from Def. 3.10.*

PROOF. Observe that F is only used on the inputs to the up and qry algorithms in Fig. 2. By identifying $\mathfrak{R} = [m]^k$ with a subset of $\mathfrak{D}$, so that formally $\mathfrak{R} \subset \mathfrak{D}$, and since $\text{Id}_{\mathfrak{R}}(F(x)) = F(x)$ for any $x \in \mathfrak{D}$, the result follows. □

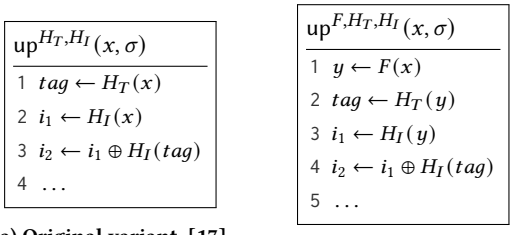
We then apply Theorem 4.2 to Bloom filters instantiated using PRFs.

COROLLARY 4.4. *Let n, q_u, q_t, q_v be non-negative integers, and let $t_a, t_d > 0$. Let $F : \mathfrak{D} \to \mathfrak{R}$. Let Π be a Bloom filter with public parameters pp and oracle access to F. If R_K for $K \leftarrow\$ \mathcal{K}$ is an $(n + q_u + q_t, t_a + t_d, \varepsilon)$-secure pseudorandom function and $F = R_K$, then Π is $(q_u, q_t, q_v, t_a, t_s, t_d, \varepsilon')$-adversarially correct, where $\varepsilon' = \varepsilon + 2q_t \cdot P_{\Pi,pp}(FP \mid n + q_u)$ and $t_s \approx t_a$.*

PROOF. From the instantiation of Bloom filters given in Fig. 2, we observe that each up and qry call contains one call to the function F. Then, using Lemma 4.3, Theorem 4.2 holds with $\alpha = \beta = 1$. □

Insertion-only Cuckoo filters. Unfortunately, Cuckoo filters are not function-decomposable. From the first few instructions of up^{H_I,H_T} (see Fig. 5a), we see that both hash functions H_I and H_T (which could be replaced with PRFs as for Bloom filters) need to be evaluated on x. If one were to attempt H_T-decomposition of up, that is, to instantiate $\text{up}^{\text{Id}_{\mathfrak{R}},H_I}(H_T(x), \sigma)$, it would not be possible to evaluate $i_1 \leftarrow H_I(x)$ from $H_T(x)$ alone. An attempt to H_I-decompose up would pose the reverse problem, having to evaluate $tag \leftarrow H_T(x)$ from only $H_I(x)$. It would also introduce a new problem, having to evaluate $i_2 \leftarrow i_1 \oplus H_I(tag)$ without access to H_I.[6] To overcome this

[6] A proof of NAI would also struggle with H_I being evaluated both on x and on tag for every call to up: it would mean that on a single call to $\text{up}(x)$, $\mathcal{A}$ would also learn $H_I(tag)$, where $tag \neq x$ with high probability.



(a) Original variant. [17] **(b) *PRF-wrapped* variant.**

Figure 5: Beginning of the up algorithm for insertion-only Cuckoo filter variants.

barrier, we propose the following minor variant of insertion-only Cuckoo filters that achieves function-decomposability and satisfies all our consistency rules, allowing it to satisfy the requirements of Theorem 4.2.

PRF-wrapped insertion-only Cuckoo filters. To address function-decomposability issues in insertion-only Cuckoo filters, we propose a generic technique: preprocessing the inputs $x \in \mathfrak{D}$ to up, qry with a random function $F\colon \mathfrak{D} \to \mathfrak{R}$, for some $\mathfrak{R} \subset \mathfrak{D}$ (including potentially $\mathfrak{R} = \mathfrak{D}$). This results in the $\mathsf{up}^{F,H_T,H_I}(x,\sigma)$ algorithm shown in Fig. 5b, and in a similarly "PRF-wrapped" qry algorithm. The resulting AMQ-PDS is easy to implement since it only requires adding a PRF call on inputs, before passing them to the existing up^{H_T,H_I} and qry^{H_T,H_I} implementations.

Lemma 4.5. *PRF-wrapped insertion-only Cuckoo filters with oracle access to a random function F are F-decomposable, reinsertion invariant, and satisfy the insertion-only AMQ-PDS consistency rules from Def. 3.10.*

Proof. This follows by inspection of the proposed modifications of the up and qry algorithms in Fig. 5b. □

As a consequence of adding a PRF computation on inputs, the NAI false positive probability is slightly increased by the probability of finding a collision of the PRF.

Lemma 4.6. *Let n be a non-negative integer, let Π be a PRF-wrapped insertion-only Cuckoo filter with public parameters $pp = (s, \lambda_I, \lambda_T, num)$, wrapped using a PRF $R_K\colon \mathfrak{D} \to \mathfrak{R}$, and let Π' be an original insertion-only Cuckoo filter with the same public parameters pp as Π's. Let $\overline{P_{\Pi,pp}}(FP \mid n) \coloneqq \overline{P_{\Pi',pp}}(FP \mid n) + \frac{(2s+2)^2}{2|\mathfrak{R}|}$. Then $P_{\Pi,pp}(FP \mid n) \leq \overline{P_{\Pi,pp}}(FP \mid n)$, where $\frac{(2s+2)^2}{2|\mathfrak{R}|}$ can be made cryptographically small.*

sketch. The result follows by repeating the analysis for original Cuckoo filters [17] (Lemma 3.9), but accounting for the chance of a collision between $2s+2$ uniformly random elements in $\mathfrak{R}$. See the full version for the proof details. □

Finally, we can apply Theorem 4.2.

Corollary 4.7. *Let n, q_u, q_t, q_v be non-negative integers, and let $t_a, t_d > 0$. Let $F : \mathfrak{D} \to \mathfrak{R}$. Let Π be a PRF-wrapped insertion-only Cuckoo filter with public parameters pp and oracle access to F. If R_K for $K \leftarrow\!\!\$\ \mathcal{K}$ is an $(n + q_u + q_t, t_a + t_d, \varepsilon)$-secure pseudorandom function and $F = R_K$, then Π is $(q_u, q_t, q_v, t_a, t_s, t_d, \varepsilon')$-adversarially correct, where $\varepsilon' = \varepsilon + 2q_t \cdot P_{\Pi,pp}(FP \mid n + q_u)$ and $t_s \approx t_a$.*

Proof. From the instantiation of PRF-wrapped Cuckoo filters (Fig. 5b), observe that each up and qry call contains one call to the function F. Applying Lemma 4.5, Theorem 4.2 holds with $\alpha = \beta = 1$. □

4.3 Discussion on correctness

We start by comparing our results with those of Clayton et al. [12]. They analysed the adversarial correctness of PDS under four deployment settings, characterised by having *private* or *public representations*, and by being either *mutable* or *immutable*. Our correctness result (Theorem 4.2) holds in all four settings by allowing the adversary up to q_v **Reveal** queries and q_u **Up** queries. The immutable setting then corresponds to $q_u = 0$ and the private setting to $q_v = 0$.

In [12], a game-based approach is used to derive bounds on the correctness of Bloom filters in the above adversarial settings. There, the adversary's win condition is to cause the event $S_r \coloneqq [\mathcal{A}$ makes $\mathbf{Qry}(\cdot)$ queries resulting in at least r false positives]. Using our simulation-based approach, we can derive new bounds for the setting of [12]. To see how, notice that extending Remark 2, Theorem 4.2 also gives a bound on the difference in probabilities of event S_r in both the real and ideal worlds. Our bound then involves a term of the form $2q_t \cdot P_{\Pi,pp}(FP \mid n + q_u)$, while the bound in [12] depends strongly on r. This should not be surprising given that our approach is general while that of [12] involves a specific winning condition posed in terms of r. While this implies we are less flexible with the value of r we implicitly tolerate, our approach covers any adversary (with no assumptions on its behaviour), illustrating the power of simulation-based notions. This includes adversaries who specify their objective not explicitly in terms of false positives caused, but perhaps in terms of a target Hamming weight of the Bloom filter's state, or in terms of subsets of the filter's index to be set to 1 (as in target-set coverage attacks [12]). Further, by not requiring a choice of r, our results do not require satisfying particular constraints on r, as in [12, Theorem 3]. In the full version, we provide an in-depth comparison of our results with those of [12], and outline how our approach covers other adversarial objectives such as target-set coverage attacks.

We note that in the immutable setting, a slight modification of our Theorem 4.2 proof gives a tighter bound with $\varepsilon' = \varepsilon$ (see the full version).

Although our proof methodology is similar in spirit to that of HLL [33], the analysis of adversarial correctness for AMQ-PDS is much more involved. The ability to make membership queries on an element gives the adversary valuable information on how useful it would be to insert that element; cardinality estimates, on the other hand, do not reveal such information.

Finally, we comment on the implications of our bounds. While the distinguishing advantage in Real-or-Ideal may not be negligible as one might expect in cryptographic proofs, by relating the adversarial setting to the well-studied honest setting from the PDS literature, we can place concrete bounds on the success of any AMQ-PDS adversary. We return to this point in § 6.

5 PRIVACY

We now shift focus to privacy guarantees for AMQ-PDS, addressing the following question: to what extent does the functionality

oracle **RepLeak**()	oracle **ElemLeak**(x)
1 **return** $\lvert V\rvert$	1 **return** $[x \in V]$

Figure 6: Leakage profile for AMQ-PDS.

of an AMQ-PDS compromise the privacy of the elements that it stores? We explore simulation-based privacy notions for AMQ-PDS. We propose two such notions for quantifying privacy, each associated with a different leakage profile in the ideal world, and investigate the relationship between the two. We identify a specific property, *permutation invariance (PI)*, as being of central importance in establishing privacy, and show that it is implied by function-decomposability of an AMQ-PDS. Finally, we apply our results to Bloom and PRF-wrapped insertion-only Cuckoo filters.

Settings. We model interactions by an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ with an AMQ-PDS Π in two stages. In the first stage, a randomised algorithm $\mathcal{A}_1$ populates Π with a set of elements $V \subset \mathfrak{D}$ via the **Rep** oracle. In the second stage, an adversary $\mathcal{A}_2$ attempts to learn something about V. We stress that no state is shared between $\mathcal{A}_1$ and $\mathcal{A}_2$ (in contrast, in § 4, the implicit two-part adversary is allowed to share state).

We consider two adversarial settings. In the *snapshot* setting, at the end of the first stage, the adversary $\mathcal{A}_2$ is given Π's state σ via access to the **Reveal** oracle, but has no other oracle access. In the *adaptive* setting, $\mathcal{A}_2$ is instead given query access to Π via the **Qry**, **Up** and **Reveal** oracles. These settings capture various real world scenarios. For example, a system intrusion might lead to the leakage of σ but go unnoticed for some time, and hence public access to Π is not disabled. Analysing privacy in the adaptive setting allows us to quantify what harm an adversary could do in such a scenario.

Leakage profiles for AMQ-PDS. To define a notion of privacy for an AMQ-PDS Π, we first need to characterise its leakage profile. This describes the information leaked as a result of Π's functionality, which also depends on whether we are in the snapshot or adaptive setting. We model the leakage as a set of functions that a simulator is allowed to use as oracles, and justify their inclusion below.

In Fig. 6 we define two leakage functions for an AMQ-PDS representing a set V of elements. The first is **RepLeak**, which leaks $|V|$. This captures the fact that for commonly used AMQ-PDS, one can estimate $|V|$ from observing σ. For example, if Π is a Bloom filter, this could be indicated by the number of bits set to 1 in σ, or if Π is a Cuckoo filter, it could be estimated from the number of tags stored in σ (up to the probability of collisions in H_T).

In settings that allow membership queries, we require a second leakage function **ElemLeak**, capturing the fact that $\mathcal{A}_2$ can always issue **Qry**(x) queries and learn their output. While this may seem to result in a weak privacy notion, such leakage is unavoidable in the real world if access to the AMQ-PDS' API is provided.

In this section, we will show that this leakage profile serves as an upper bound of the real leakage of function-decomposable AMQ-PDS, by constructing simulators that use it to provide consistent views of AMQ-PDS to adversaries.

5.1 Notions of Privacy: Elem-Rep privacy

We start with a high-level explanation of our first privacy definition, *Elem-Rep privacy*. We again employ a simulation-based approach. Let the AMQ-PDS be populated with elements from a set V by adversary $\mathcal{A}_1$. Then, adversary $\mathcal{A}_2$ interacts with the AMQ-PDS through the setting-specific oracles. Adversary $\mathcal{A}_2$ plays in either a real or ideal world. In the real world, it interacts with a keyed AMQ-PDS initialised with the elements in V. In the ideal world, it interacts with a simulator $\mathcal{S}$. The simulator does not know V, but has access to **RepLeak** in the snapshot setting, and additionally **ElemLeak** in the adaptive setting. The output of $\mathcal{A}_2$ is then given to a distinguisher $\mathcal{D}$ along with V. By showing that the distinguisher's outputs in the real and ideal world are close, we prove that the adversary cannot learn much more about the elements in V through interacting with the AMQ-PDS in the real world than in the ideal world (where it can only learn the specified leakage from $\mathcal{S}$).

We formalise the above in Fig. 7, in the R-or-I-ElemRepPriv game. We will use *Real* and *Ideal* to denote the real (d=0) and ideal (d=1) versions of the R-or-I-ElemRepPriv game. As in § 4, if an oracle **O** is not directly specified, we will assume it is defined as in Fig. 3.

Definition 5.1 (Elem-Rep privacy). Let Π be an insertion-only AMQ-PDS with public parameters pp, and let R_K be a keyed function family. Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be a tuple of algorithms. We say Π is $(q_u, q_t, q_v, t_a, t_s, t_d, \varepsilon)$-Elem-Rep private if there exists a simulator $\mathcal{S}$ (that runs $\mathcal{A}_2$, calls oracle **RepLeak** at most once, calls oracle **ElemLeak** only when $\mathcal{A}_2$ calls its **Qry**, **Up** oracles and on the same argument, and runs in time at most t_s) such that, for all $\mathcal{A}_1$, for all $\mathcal{A}_2$ running in time at most t_a and making q_u, q_t, q_v queries to oracles **Up**, **Qry**, **Reveal** respectively, and for all distinguishers $\mathcal{D}$ running in time at most t_d, we have:

$$\mathrm{Adv}^{RoIElemRepP}_{\Pi,\mathcal{A},\mathcal{S}}(\mathcal{D}) := \left|\Pr[Real(\mathcal{A},\mathcal{D})=1] - \Pr[Ideal(\mathcal{A},\mathcal{D},\mathcal{S})=1]\right| \leq \varepsilon,$$

in R-or-I-ElemRepPriv (Fig. 7).

Informally, Def. 5.1 implies that the API of an Elem-Rep private AMQ-PDS Π does not leak more than the number of elements in Π and the true query responses for elements queried via **Up** and **Qry**.

In the following sections, we show how to compute bounds on the Elem-Rep privacy of an AMQ-PDS. We start by introducing a property that we call *permutation invariance* (PI), show that it is implied by function-decomposability, and that in turn it implies a bound on Elem-Rep privacy of an AMQ-PDS.

5.2 Permutation invariance (PI)

Consider the Exp^{PI}_{Π} game in Fig. 8, where an adversary $\mathcal{A}$, who has access to the **Rep**, **Up**, **Qry**, **Reveal** oracles, must distinguish between an AMQ-PDS where the inputs to all queries are either randomly permuted or not.

Definition 5.2. Let Π be an insertion-only AMQ-PDS, with public parameters pp. We say Π is $(q_u, q_t, q_v, t_a, \varepsilon)$-permutation invariant ($\varepsilon$-PI for short, 0-PI when $\varepsilon = 0$) if, for all adversaries $\mathcal{B}$ running in time at most t_a and making first a single query to **Rep** and then q_u, q_t, q_v queries to oracles **Up**, **Qry**, **Reveal** respectively, we have:

$$\mathrm{Adv}^{PI}_{\Pi}(\mathcal{B}) := \left|\Pr[Exp^{PI}_{\Pi}(\mathcal{B})=1 \mid c=0] - \Pr[Exp^{PI}_{\Pi}(\mathcal{B})=1 \mid c=1]\right| \leq \varepsilon,$$

R-or-I-ElemRepPriv($\mathcal{A}, \mathcal{S}, \mathcal{D}, pp$)

1 $d \leftarrow\$ \{0,1\}$
2 init $\leftarrow \bot$
3 $V \leftarrow\$ \mathcal{A}_1$
4 **if** $d = 0$ // Real
5 $K \leftarrow\$ \mathcal{K}; F \leftarrow R_K$
6 $\sigma \leftarrow \text{setup}(pp)$
7 **Rep**(V)
8 $out \leftarrow\$ \mathcal{A}_2^{\boxed{\textbf{Up, Qry,}}\ \textbf{Reveal}}$
9 **else** // Ideal
10 $out \leftarrow\$ \mathcal{S}^{\boxed{\textbf{ElemLeak,}}\ \textbf{RepLeak}}(\mathcal{A}_2, pp)$
11 $d' \leftarrow\$ \mathcal{D}(out, V)$
12 **return** d'

R-or-I-RepPriv($\mathcal{A}, \mathcal{S}, \mathcal{D}, pp$)

1 $d \leftarrow\$ \{0,1\}$
2 init $\leftarrow \bot$
3 $V \leftarrow\$ \mathcal{A}_1$
4 **if** $d = 0$ // $\overline{\text{Real}}$
5 $K \leftarrow\$ \mathcal{K}$; $F \leftarrow R_K$
6 $\sigma \leftarrow \text{setup}(pp)$
7 **for** $x \in V$
8 $(b, \sigma) \leftarrow\$ \text{up}^F(x, \sigma)$
9 init $\leftarrow \top$
10 $out \leftarrow\$ \mathcal{A}_2^{\boxed{\textbf{Up, Qry,}}\ \textbf{Reveal}}$
11 **else** // $\overline{\text{Ideal}}$
12 $out \leftarrow\$ \mathcal{S}^{\textbf{RepLeak}}(\mathcal{A}_2, pp)$
13 **return** $d' \leftarrow\$ \mathcal{D}(out, V)$

Figure 7: Elem-Rep (resp. Rep) privacy game for AMQ-PDS Π, with respect to the leakage profile obtained by combining RepLeak and ElemLeak (resp. the leakage profile consisting only of RepLeak), in the snapshot and adaptive settings.

$Exp_{\Pi}^{PI}(\mathcal{B})$

1 $F \leftarrow\$ \text{Funcs}[\mathfrak{D}, \mathfrak{R}]$
2 init $\leftarrow \bot$
3 $\sigma \leftarrow \text{setup}(pp)$
4 $c \leftarrow\$ \{0,1\}$
5 **if** $c = 1$
6 $\pi \leftarrow\$ \text{Perms}[\mathfrak{D}]$
7 **else** : $\pi \leftarrow \text{Id}_{\mathfrak{D}}$
8 $c' \leftarrow\$ \mathcal{B}^{\textbf{Rep},\boxed{\textbf{Up, Qry,}}\ \textbf{Reveal}}$
9 **return** c'

Oracle **Rep**(V)

1 **if** init $= \top$: **return** $\bot$
2 init $\leftarrow \top$
3 **for** $x \in V$
4 $(b, \sigma) \leftarrow\$ \text{up}^F(\pi(x), \sigma)$
5 **return** $\top$

Oracle **Up**(x)

1 **if** init $= \bot$: **return** $\bot$
2 $(b, \sigma) \leftarrow\$ \text{up}^F(\pi(x), \sigma)$
3 **return** b

Oracle **Qry**(x)

1 **if** init $= \bot$
2 **return** $\bot$
3 $a \leftarrow \text{qry}^F(\pi(x), \sigma)$
4 **return** a

Oracle **Reveal**()

1 **return** σ

Figure 8: PI game for AMQ-PDS Π in the snapshot and adaptive settings.

in Exp_{Π}^{PI} (Fig. 8). We say $\mathcal{B}$ is a (q_u, q_t, q_v, t_a)-PI adversary.

Our next result relates function-decomposability of an AMQ-PDS (as per Def. 3.1) to its permutation invariance.

Lemma 5.3. *Let $F \leftarrow\$ \text{Funcs}[\mathfrak{D}, \mathfrak{R}]$ be a random function, and let Π be an F-decomposable AMQ-PDS with public parameters pp and oracle access to F. Then Π is 0-PI.*

sketch. By F-decomposability, we rewrite $\text{up}^F(\pi(x), \sigma)$ calls in Exp_{Π}^{PI} as $\text{up}^{\text{Id}_{\mathfrak{R}}}(F(\pi(x)), \sigma)$, and similarly for qry. Since $F(x)$ and $F(\pi(x))$ are random functions, the $c = 0, 1$ versions of Exp_{Π}^{PI} are indistinguishable. A full proof is given in the full version. □

Similarly to correctness (§ 4.1), Lemma 5.3 only refers to a single oracle function F, but it also holds for AMQ-PDS using t oracle functions $F_1, \ldots F_t$ and being F_1-decomposable.

Simulator $\mathcal{S}_{\text{Elem-Rep}}(\mathcal{A}_2, pp)$

1 $F \leftarrow\$ \text{Funcs}[\mathfrak{D}, \mathfrak{R}]$
2 $\sigma \leftarrow \text{setup}(pp)$
3 $P \leftarrow \{\}$ // key-value store
4 $Y \leftarrow \{\}$ // set
5 **RepSim**()
6 $out \leftarrow\$ \mathcal{A}_2^{\textbf{UpSim,QrySim,RevSim}}$
7 **return** out

Procedure **RepSim**()

1 $n' \leftarrow$ **RepLeak**()
2 **for** $i \in [n']$
3 $y \leftarrow\$ \mathfrak{D} \setminus Y$
4 $Y \leftarrow Y \cup \{y\}$
5 $(b, \sigma) \leftarrow\$ \text{up}^F(y, \sigma)$
6 **return** $\top$

Procedure **Per**(x)

1 **if** $P[x] = \bot$
2 **if ElemLeak**$(x) = \top$
3 $P[x] \leftarrow\$ Y \setminus \text{vals}(P)$
4 **else**
5 $P[x] \leftarrow\$ \mathfrak{D} \setminus (\text{vals}(P) \cup Y)$
6 **return** $P[x]$

Oracle **UpSim**(x)

1 $(b, \sigma) \leftarrow\$ \text{up}^F(\textbf{Per}(x), \sigma)$
2 **return** b

Oracle **QrySim**(x)

1 **return** $\text{qry}^F(\textbf{Per}(x), \sigma)$

Oracle **RevSim**()

1 **return** σ

Figure 9: Simulator used to prove Theorem 5.4.

5.3 Elem-Rep privacy from PI

We now show how to compute a bound on the Elem-Rep privacy of an AMQ-PDS using permutation invariance.

Theorem 5.4. *Let q_u, q_t, q_v be non-negative integers, and let t_a, $t_d > 0$. Let $F : \mathfrak{D} \to \mathfrak{R}$. Let Π be an insertion-only AMQ-PDS with public parameters pp and oracle access to F. Let n be the maximum number of elements returned by $\mathcal{A}_1$ on line 3 of R-or-I-ElemRepPriv in Fig. 7. Let α (resp. β) be the number of calls to F required to insert (resp. query) an element in Π using its up (resp. qry) algorithm.*

If $F \equiv R_K : \mathfrak{D} \to \mathfrak{R}$ is an $(\alpha(n + q_u) + \beta q_t, t_a + t_d, \varepsilon_{PRF})$-secure pseudorandom function with key $K \leftarrow\$ \mathcal{K}$, and Π is $(q_u, q_t, q_v, t_a, \varepsilon_{PI})$-permutation invariant, then Π is $(q_u, q_t, q_v, t_a, t_s, t_d, \varepsilon_{PRF} + \varepsilon_{PI})$-Elem-Rep private, where $t_s \approx t_a$.

sketch. We construct a simulator $\mathcal{S}_{\text{Elem-Rep}}$ for *Ideal* in Fig. 9, and an adversary $\mathcal{B}$ in Exp_{Π}^{PI} that runs the R-or-I-ElemRepPriv adversary $\mathcal{A}$ internally. Then, we rewrite $\mathcal{A}$'s advantage in terms of the permutation invariance of Π, the distance between *Real* and the $c = 0$ version of Exp_{Π}^{PI}, and that of *Ideal* and the $c = 1$ version of Exp_{Π}^{PI}. The first (resp. second) term is bounded by ε_{PI} (resp. ε_{PRF}). Observing that *Ideal* is indistinguishable from the $c = 1$ version of Exp_{Π}^{PI} gives the result. The full proof is given in the full version. □

This result essentially tells us that, up to the $\varepsilon_{\text{PRF}} + \varepsilon_{\text{PI}}$ bound, the state of a function-decomposable AMQ-PDS representing some set V does not leak more than the number of elements in the data structure, and that querying the AMQ-PDS does not reveal more than the true answers to set membership queries of queried elements. However, such a guarantee may not be useful for computing concrete bounds on privacy in practice, as it does not explicitly quantify the impact of what **ElemLeak** reveals or how this is related to the distribution of elements in V. This motivates our alternative privacy notion, introduced in the next section.

5.4 Notions of Privacy: Rep privacy

In this section, we define a second privacy notion, *Rep privacy*. Here, the simulator no longer has access to the **ElemLeak** oracle, but instead only to the **RepLeak** oracle, allowing it to learn $|V|$ where $V \leftarrow\!\!\$\ \mathcal{A}_1$. We formalise this privacy definition by the R-or-I-RepPriv game in Fig. 7. We will use $\overline{Real}, \overline{Ideal}$ to denote the real ($d = 0$), ideal ($d = 1$) versions of the game, respectively.

Definition 5.5 (Rep privacy). Let Π be an insertion-only AMQ-PDS with public parameters pp, and let R_K be a keyed function family. Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be a tuple of algorithms. We say Π is $(q_u, q_t, q_v, t_a, t_s, t_d, \varepsilon)$-Rep private if there exists a simulator $\mathcal{S}$ (that runs $\mathcal{A}_2$, calls **RepLeak** at most once, and runs in time at most t_s) such that, for all $\mathcal{A}_1$, for all $\mathcal{A}_2$ running in time at most t_a and making q_u, q_t, q_v queries to oracles **Up**, **Qry**, **Reveal** respectively, and for all distinguishers $\mathcal{D}$ running in time at most t_d, we have:

$$\mathrm{Adv}^{RoIRepP}_{\Pi,\mathcal{A},\mathcal{S}}(\mathcal{D}) := \left|\Pr[\overline{Real}(\mathcal{A},\mathcal{D})=1] - \Pr[\overline{Ideal}(\mathcal{A},\mathcal{D},\mathcal{S})=1]\right| \leq \varepsilon,$$

in R-or-I-RepPriv (Fig. 7).

While the notion of Rep privacy is exactly the same as Elem-Rep privacy in the snapshot setting, their relationship is more subtle in the adaptive setting. By removing the simulator's access to **ElemLeak**, the bound obtained through Rep privacy is directly related to the probability of guessing elements in V. In fact, we show this formally in the following theorem.

Theorem 5.6. *Let q_u, q_t, q_v be non-negative integers, and let $t_a, t_d > 0$. Let Π be an insertion-only AMQ-PDS, with public parameters pp. Suppose Π is $(q_u, q_t, q_v, t_a, t_s, t_d, \varepsilon)$-Elem-Rep private with simulator $\mathcal{S}$. Then there exists a simulator $\mathcal{S}'$ (that is constructed from $\mathcal{S}$) such that Π is $(q_u, q_t, q_v, t_a, t_{s'}, t_d, \varepsilon')$-Rep private with simulator $\mathcal{S}'$ and $t_s \approx t_{s'}$. Here $\varepsilon' = \varepsilon + \Pr[W \cap V \neq \emptyset]$, with W denoting the set of elements queried by $\mathcal{A}_2$ to its **Up**, **Qry** oracles in $\overline{Ideal}$ within $\mathcal{S}'$.*

sketch. Our goal is to relate $\overline{Real}, \overline{Ideal}$ from R-or-I-RepPriv to *Real*, *Ideal* from R-or-I-ElemRepPriv. We first construct a simulator $\mathcal{S}'$ that is the same as $\mathcal{S}$, but where every **ElemLeak** call is replaced with $\perp$. Then, *Ideal* and $\overline{Ideal}$ (the worlds simulated by $\mathcal{S}$ and $\mathcal{S}'$, respectively) are identical up until $\mathcal{A}_2$ makes an **Up** or **Qry** query on something in V, which occurs with probability $\Pr[W \cap V \neq \emptyset]$. Since *Real* and $\overline{Real}$ are identical, we use the ε-Elem-Rep privacy of Π to obtain the result. See the full version for the full proof. □

While Theorems 5.4 and 5.6 only refer to a single oracle function F, the same result holds also for AMQ-PDS using t oracle functions $F_1, \ldots F_t$ and being F_1-decomposable.

5.5 Guarantees for Bloom and Cuckoo filters

Using our analysis, we derive results on the privacy of Bloom filters.

Corollary 5.7. *Let n, q_u, q_t, q_v be non-negative integers, and let $t_a, t_d > 0$. Let $F : \mathfrak{D} \to \mathfrak{R}$. Let Π be a Bloom filter with public parameters pp and oracle access to F. Let δ denote $\Pr[\, W \cap V \neq \emptyset \,]$. If R_K for $K \leftarrow\!\!\$\ \mathcal{K}$ is an $(n + q_u + q_t, t_a + t_d, \varepsilon)$-secure PRF and $F = R_K$, then Π is $(q_u, q_t, q_v, t_a, t_s, t_d, \varepsilon + \delta)$-Rep private, where $t_s \approx t_a$.*

Proof. Since Π is F-decomposable (Lemma 4.3), it satisfies 0-PI (Lemma 5.3). We then set $\varepsilon_{\mathrm{PI}} = 0$, $\varepsilon_{\mathrm{PRF}} = \varepsilon$ in Theorem 5.4 with $\alpha = \beta = 1$ to obtain an Elem-Rep privacy bound, and finally apply Theorem 5.6 to convert this to a Rep privacy bound. □

We can prove similar results for the Cuckoo filter.

Corollary 5.8. *Let n, q_u, q_t, q_v be non-negative integers, and let $t_a, t_d > 0$. Let $F : \mathfrak{D} \to \mathfrak{R}$. Let Π be a PRF-wrapped insertion-only Cuckoo filter with public parameters pp and oracle access to F. Let δ denote $\Pr[\, W \cap V \neq \emptyset \,]$. If R_K for $K \leftarrow\!\!\$\ \mathcal{K}$ is an $(n + q_u + q_t, t_a + t_d, \varepsilon)$-secure pseudorandom function and $F = R_K$, then Π is $(q_u, q_t, q_v, t_a, t_s, t_d, \varepsilon + \delta)$-Rep private, where $t_s \approx t_a$.*

sketch. The proof proceeds similarly to Corollary 5.7, using Lemmas 4.5 and 5.3 along with Theorems 5.4 and 5.6. □

We note that our PRF-wrapped variant of the insertion-only Cuckoo filter simplifies not only our correctness analysis but also privacy, by attaining function-decomposability. Furthermore, the original insertion-only Cuckoo filter (Fig. 5a) does not satisfy 0-PI due to a trivial distinguishing attack (see full version).

5.6 Discussion on privacy

We explored two ways of defining a simulation-based privacy notion for AMQ-PDS, each with respect to a specific leakage profile. Our first privacy definition, Elem-Rep privacy, has a leakage profile capturing the information intrinsically leaked by the AMQ-PDS about the elements it stores. However, the bound obtained by quantifying the Elem-Rep privacy is not trivial to interpret; one must also carefully analyse the leakage to determine the amount of privacy obtained in practice, as is common in simulation-based notions. Our alternative definition, Rep privacy, has a smaller and simpler leakage profile, but the bound explicitly depends on how easy it is to guess elements stored in the AMQ-PDS. This approach may be more useful in practice; for example, it allows to directly relate privacy to the min-entropy of the distribution of stored elements.

Our results confirm the intuition that one cannot hope to achieve privacy if the elements stored in the AMQ-PDS are easy to predict, a "low min-entropy" scenario. In this setting, **ElemLeak** would reveal a substantial amount of information as the adversary is likely to query elements in V, leading to the Elem-Rep privacy bound being a weak guarantee in practice. Similarly, we would not obtain a good bound via Rep privacy either, as the term $\Pr[\, W \cap V \neq \emptyset \,]$ would be high. On the other hand, a "high min-entropy" scenario results in strong guarantees from both privacy notions. In fact, the number of **ElemLeak** queries that give the adversary useful information is directly linked to its likelihood of guessing elements in V.

We note that our privacy theorems can be used to analyse various real-world scenarios. For example, by setting $q_u = 0$ in Theorems 5.4 and 5.6, we cover the "static data" scenario, where an application first adds a set of elements to a PDS, and the adversary's goal is to learn these elements through only set membership queries.

Throughout this section, we have assumed that leaking $|V|$ is acceptable, as this is unavoidable for Bloom and Cuckoo filters in the *public* setting (i.e. when the **Reveal** oracle is available). However, there may be settings where $|V|$ is sensitive, in which case one may want to investigate alternative AMQ-PDS (see full version).

6 SECURE INSTANCES

We sketch how to use our results to instantiate AMQ-PDS instances achieving provable guarantees. An expanded discussion is provided in the full version, covering both correctness and privacy. Here we focus on the former aspect. We aim to use our bounds to set AMQ-PDS parameters. Recall the guarantee from Theorem 4.2:

$$\mathrm{Adv}^{\mathrm{RoI}}_{\Pi,\mathcal{A},\mathcal{S}}(\mathcal{D}) = |\Pr[\mathit{Real}(\mathcal{A},\mathcal{D})=1] - \Pr[\mathit{Ideal}(\mathcal{A},\mathcal{D},\mathcal{S})=1]|$$
$$= |\Pr[\mathcal{D}(\mathcal{A})=1] - \Pr[\mathcal{D}(\mathcal{S}(\mathcal{A},pp))=1]| \leq \varepsilon + 2q_t \cdot P_{\Pi,pp}(FP \mid n+q_u),$$

where ε is a PRF distinguishing advantage, n is the number of elements initially inserted into the AMQ-PDS, and $q := q_u + q_t$ is the total number of queries made by $\mathcal{A}$ that influence its success probability. Crucially, usually $P_{\Pi,pp}(FP \mid n+q_u)$ can be estimated using well-established upper bounds, cf. Lemmas 3.7 and 4.6.

This result allows us to establish an upper bound on the probability $\Pr[\mathcal{D}(\mathcal{A})=1]$ of an adversary $\mathcal{A}$ with the given query budget q finding a sequence of queries to an AMQ-PDS Π that allows them to satisfy some desired predicate P in the *Real* world, by relating this probability to that of $\mathcal{A}$ satisfying P in the *Ideal* world. As a practical example, we investigate the choice of adversarially correct parameters for Bloom and PRF-wrapped Cuckoo filters for one possible query budget; two more budgets are explored in the full version. Concretely, we bound the probability that after $n + q_u$ adversarial insertions, querying a random non-inserted element returns a false positive result, cf. Remark 2. In Fig. 10 we plot an upper bound of the false positive probability against the size of the data structure in both adversarial and non-adversarial settings for various public parameters. Our results show that achieving protection against adversarial inputs requires roughly doubling (Bloom) or trebling (insertion-only Cuckoo) the storage used, as compared to the honest setting.

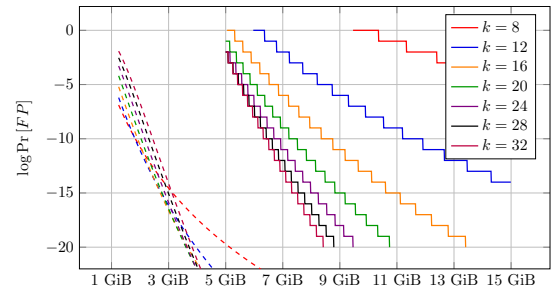
Remark 3. *We stress that to obtain correctness and privacy, the key to any PRFs used needs to be stored securely, say in hardware, so as to resist being exposed by a state reveal.*

7 CONCLUSIONS

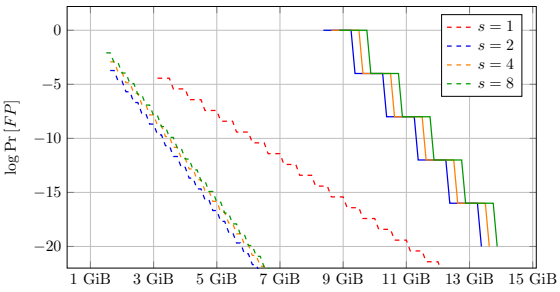
We have introduced a framework for analysing the correctness (under adversarial input) and privacy of AMQ-PDS. We employed a simulation-based approach, with correctness and privacy emerging through imposing different constraints on the simulators. We have applied our approach to study Bloom and insertion-only Cuckoo filters, showing how they may be securely instantiated and highlighting the cost of adding security over unprotected instances.

Our work lays a foundation for further study and analysis. Some important topics for future investigation include: • How tight are the bounds we provide? Are there matching attacks or can tighter bounds be proven? This is important since using our bounds to set concrete AMQ-PDS parameters incurs overhead in storage. • Can our Cuckoo filter analysis be extended to allow deletion of elements? This would require extension of our syntax and models, as well as careful modification of the consistency rules. • Can our simulation-based approach be extended to other classes of PDS? Does "wrapping" the inputs of a PDS using a PRF always work as a protection? Investigating this would require developing a general syntax for PDS beyond that in our work and in [12]. • We focused on a strong adversarial model, where the adversary can access the AMQ-PDS's state and has full adaptivity in its queries, necessitating the use of PRFs to achieve security. Do weaker primitives, e.g. UOWHFs, suffice in weaker settings, such as if the state is not accessible to the adversary? For example, the results of [12] suggest that salted hashes may indeed suffice. • While we considered an adversary attacking an honest service provider via an API, one may also ask: how can a user of the API obtain guarantees about the accuracy of the service being provided? • How are our simulation-based correctness notions and the game-based ones in [12] related? Can we show a form of equivalence akin to that between semantic and IND-CPA security?

We conclude by remarking that cryptographic tools and thinking seem to be broadly applicable to the problem of understanding the behaviour of PDS in adversarial settings. This topic is currently relatively under-researched, but is of growing importance in view of how rapidly PDS are being adopted in practice.



(a) Bloom, $\log n = 7$, $\log(q_u + q_t) = 30$



(b) Cuckoo, $\log n = 7$, $\log(q_u + q_t) = 30$

Figure 10: Correctness guarantees vs. storage trade-offs for Bloom and PRF-wrapped insertion-only Cuckoo filters. Solid lines represent adversarial guarantees ($\log \Pr[FP] \geq \log \Pr[\mathcal{D}(\mathcal{A}) = 1]$**). Dashed lines represent the values obtained assuming NAI (**$\log \Pr[FP] = \log \overline{P_{\Pi,pp}}(FP \mid n + q_u)$**).**

ACKNOWLEDGMENTS

The work of Filić was supported by the Microsoft Research Swiss Joint Research Center. The work of Paterson was supported in part by a gift from VMware. The work of Virdia was supported by the Zurich Information Security and Privacy Center.

REFERENCES

[1] Noga Alon, Yossi Matias, and Mario Szegedy. 1996. The Space Complexity of Approximating the Frequency Moments. In *Proceedings of ACM STOC'96*, Gary L.

Miller (Ed.). https://doi.org/10.1145/237814.237823
[2] Jean-Philippe Aumasson and Daniel J Bernstein. 2012. SipHash: a fast short-input PRF. In *INDOCRYPT*. Springer.
[3] Steven M. Bellovin and William R. Cheswick. 2004. Privacy-Enhanced Searches Using Encrypted Bloom Filters. Cryptology ePrint Archive, Report 2004/022. (2004).
[4] Omri Ben-Eliezer, Rajesh Jayaram, David P. Woodruff, and Eylon Yogev. 2020. A Framework for Adversarially Robust Streaming Algorithms. In *Proceedings of ACM PODS*. https://doi.org/10.1145/3375395.3387658
[5] Giuseppe Bianchi, Lorenzo Bracciale, and Pierpaolo Loreti. 2012. "Better Than Nothing" Privacy with Bloom Filters: To What Extent?. In *Proceedings of Privacy in Statistical Databases*. https://doi.org/10.1007/978-3-642-33627-0_27
[6] Burton H Bloom. 1970. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM* 13, 7 (1970), 422–426.
[7] Dan Boneh, Eyal Kushilevitz, Rafail Ostrovsky, and William E Skeith. 2007. Public key encryption that allows PIR queries. In *CRYPTO*.
[8] Novak Boskov, Ari Trachtenberg, and David Starobinski. 2020. Birdwatching: False Negatives In Cuckoo Filters. In *Proceedings of the Student Workshop (CoNEXT'20)*. 2. https://doi.org/10.1145/3426746.3434063
[9] Alexander Dodd Breslow and Nuwan Jayasena. 2018. Morton Filters: Faster, Space-Efficient Cuckoo Filters via Biasing, Compression, and Decoupled Logical Sparsity. *Proc. VLDB Endow.* (2018). https://doi.org/10.14778/3213880.3213884
[10] Andrei Z. Broder and Michael Mitzenmacher. 2003. Survey: Network Applications of Bloom Filters: A Survey. *Internet Math.* 1, 4 (2003), 485–509. https://doi.org/10.1080/15427951.2004.10129096
[11] CERT Coordination Center. 2011. Hash table implementations vulnerable to algorithmic complexity attacks. https://www.kb.cert.org/vuls/id/903934. (December 2011).
[12] David Clayton, Christopher Patton, and Thomas Shrimpton. 2019. Probabilistic data structures in adversarial environments. In *ACM SIGSAC CCS*.
[13] Graham Cormode. 2008. *Count-Min Sketch*. Springer US, 1–6. https://doi.org/10.1007/978-3-642-27848-8_579-1
[14] Graham Cormode and S. Muthukrishnan. 2005. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms* 55, 1 (2005), 58–75. https://doi.org/10.1016/j.jalgor.2003.12.001
[15] Scott A Crosby and Dan S Wallach. 2003. Denial of Service via Algorithmic Complexity Attacks.. In *USENIX Security Symposium*. 29–44.
[16] Damien Desfontaines, Andreas Lochbihler, and David Basin. 2019. Cardinality Estimators do not Preserve Privacy. In *Proceedings of PETS*.
[17] Bin Fan, Dave G. Andersen, Michael Kaminsky, and Michael D. Mitzenmacher. 2014. Cuckoo Filter: Practically Better Than Bloom. In *Proceedings of CoNEXT '14*. https://doi.org/10.1145/2674005.2674994
[18] Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier. 2007. Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. In *Discrete Mathematics and Theoretical Computer Science*. 137–156.
[19] Thomas Gerbet, Amrit Kumar, and Cédric Lauradoux. 2015. The Power of Evil Choices in Bloom Filters. In *Proceedings IEEE/IFIP DSN '15*. https://doi.org/10.1109/DSN.2015.21
[20] Arthur Gervais, Srdjan Capkun, Ghassan O. Karame, and Damian Gruber. 2014. On the privacy provisions of Bloom filters in lightweight bitcoin clients. In *Proceedings ACSAC 2014*. https://doi.org/10.1145/2664243.2664267
[21] Ashish Goel and Pankaj Gupta. 2010. Small Subset Queries and Bloom Filters Using Ternary Associative Memories, with Applications. In *Proceedings of ACM SIGMETRICS*. https://doi.org/10.1145/1811039.1811056
[22] Eu-Jin Goh. 2003. Secure Indexes. Cryptology ePrint Archive, Report 2003/216. (2003).
[23] Avinatan Hassidim, Haim Kaplan, Yishay Mansour, Yossi Matias, and Uri Stemmer. 2020. Adversarially Robust Streaming Algorithms via Differential Privacy. In *NeurIPS 2020*. https://proceedings.neurips.cc/paper/2020/hash/0172d289da48c48de8c5ebf3de9f7ee1-Abstract.html
[24] Haim Kaplan, Yishay Mansour, Kobbi Nissim, and Uri Stemmer. 2021. Separating adaptive streaming from oblivious streaming using the bounded storage model. In *CRYPTO*.
[25] Mehmet Kuzu, Murat Kantarcioglu, Elizabeth Durham, and Bradley Malin. 2011. A constraint satisfaction cryptanalysis of Bloom filters in private record linkage. In *PETS*. Springer.
[26] James Larisch, David Choffnes, Dave Levin, Bruce M Maggs, Alan Mislove, and Christo Wilson. 2017. CRLite: A scalable system for pushing all TLS revocations to all browsers. In *IEEE S&P*.
[27] Yehuda Lindell. 2016. How To Simulate It – A Tutorial on the Simulation Proof Technique. Cryptology ePrint Archive, Report 2016/046. (2016).
[28] Richard J. Lipton and Jeffrey F. Naughton. 1993. Clocked Adversaries for Hashing. *Algorithmica* 9, 3 (1993), 239–252. https://doi.org/10.1007/BF01190898
[29] Páll Melsted and Jonathan K Pritchard. 2011. Efficient counting of k-mers in DNA sequences using a bloom filter. *BMC Bioinformatics* 12 (2011).
[30] Moni Naor and Eylon Yogev. 2015. Bloom filters in adversarial environments. In *CRYPTO*.
[31] Moni Naor and Eylon Yogev. 2019. Bloom Filters in Adversarial Environments. *ACM Trans. Algorithms* 15, 3 (2019), 35:1–35:30. https://doi.org/10.1145/3306193
[32] Ryo Nojima and Youki Kadobayashi. 2009. Cryptographically Secure BloomFilters. *Trans. Data Priv.* 2, 2 (2009), 131–139. http://www.tdp.cat/issues/abs.a015a09.php
[33] Kenneth G. Paterson and Mathilde Raynal. 2022. HyperLogLog: Exponentially Bad in Adversarial Settings. In *7th IEEE European Symposium on Security and Privacy, EuroS&P 2022, Genoa, Italy, June 6-10, 2022*. IEEE, 154–170. https://doi.org/10.1109/EuroSP53844.2022.00018
[34] Henning Perl, Yassene Mohammed, Michael Brenner, and Matthew Smith. 2012. Fast confidential search for bio-medical data using Bloom filters and Homomorphic Cryptography. *2012 IEEE 8th International Conference on E-Science* (2012), 1–8.
[35] Mariana Raykova, Binh Vo, Steven M. Bellovin, and Tal Malkin. 2009. Secure anonymous database search. In *Proceedings of ACM workshop on Cloud computing security*, Radu Sion and Dawn Song (Eds.). https://doi.org/10.1145/1655008.1655025
[36] David P. Woodruff and Samson Zhou. 2020. Tight Bounds for Adversarially Robust Streams and Sliding Windows via Difference Estimators. *CoRR* abs/2011.07471 (2020). arXiv:2011.07471 https://arxiv.org/abs/2011.07471