A note on securing insertion-only Cuckoo filters

Fernando Virdia[1] Mia Filić[2],

[1] NOVA LINCS & Universidade NOVA de Lisboa, Lisbon, Portugal
[2] Applied Cryptography Group, ETH Zürich, ETH Zürich, Switzerland

Abstract. We describe a small tweak to Cuckoo filters that allows securing them under insertions using the techniques from Filić *et al.* (ACM CCS 2022), without the need for an outer PRF call.

In [FPUV22], Filić, Paterson, Unnikrishnan and Virdia define simulation-based security notions that capture some privacy and correctness guarantees for Probabilistic Data Structures handling Approximate Membership Queries (AMQ-PDS). They proceed to prove that Bloom filters [Blo70] trivially achieve these guarantees when the hash functions used by the filter are replaced with pseudorandom functions (PRF). They then attempt to prove the same for "insertion-only" Cuckoo filters [FAKM14] (*i.e.*, Cuckoo filters where only the insertion functionality is made available to the adversary) but notice that simply replacing hash functions with PRFs is not sufficient in this case. They address this issue by pre-processing inputs to the Cuckoo filter by first passing them through a PRF.

In formal terms, the issue with Cuckoo filters (solved by adding an outer PRF call) is their lack of *function-decomposability* (*c.f.* Definition 3.1 and Section 4.2 of [FPUV22]). Here, we propose a different tweak to Cuckoo filters that provides them with function-decomposability without adding an outer PRF call on top of the two original hash-function calls. In the following, we use the same notation used by Filić *et al.* First, we recall the description of Cuckoo filters given by [FPUV22].

Definition 1 (Cuckoo filter). *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 Figure 1, 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 order to achieve function-decomposability without pre-processing inputs, we increase the output length of H_T by λ_I bits, such that $H_T\colon \mathfrak{D} \to \{0,1\}^{\lambda_T+\lambda_I}$. We then read from the output of H_T (rather than from the output of H_I) the index i_1 of the first candidate bucket used to "store" $x \in \mathfrak{D}$, and use H_I only to derive the index i_2 of the second candidate bucket. This would mean defining qry^{H_T,H_I} and up^{H_T,H_I} as shown in Figure 2.

Definition 2 (Function-decomposable Cuckoo filter). *Let $pp = (s, \lambda_I, \lambda_T, num)$ be a tuple of positive integers. We define an $(s, \lambda_I, \lambda_T, num)$-function-decomposable Cuckoo filter to be the AMQ-PDS with algorithms obtained by applying the changes in Figure 2 to Figure 1, making use of hash functions $H_T\colon \mathfrak{D} \to \{0,1\}^{\lambda_T+\lambda_I}$ and $H_I\colon \mathfrak{D} \to \{0,1\}^{\lambda_I}$.*

Lemma 1. *Function-decomposable Cuckoo filters from Definition 2 with oracle access to a random function F are F-decomposable [FPUV22, Def. 3.1], reinsertion invariant [FPUV22, Def. 3.2], and satisfy consistency rules of insertion-only AMQ-PDS [FPUV22, Def. 3.10].*

Proof. Let $F \leftarrow\!\!\$\ \mathsf{Funcs}[\mathfrak{D}, \mathfrak{R}]$ where $\mathfrak{R} = \{0,1\}^{\lambda_T+\lambda_I} \subset \mathfrak{D}$. Replace H_T with F in Definition 2. Then, F-decomposability follows from observing that

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

where $\mathsf{up}^{\mathrm{Id}_{\mathfrak{R}},H_I}$ and $\mathsf{qry}^{\mathrm{Id}_{\mathfrak{R}},H_I}$ lack oracle access to F, which is truly random. Reinsertion invariance and the other consistency properties follow from inspection of up^{F,H_I} and qry^{F,H_I}, in the same way they hold for Cuckoo filters.

setup(pp)

1 $s, \lambda_I, \lambda_T, num \leftarrow pp$
2 `// Initialise` 2^{λ_I} `buckets,` s λ_T`-bit slots`
3 **for** $i \in 2^{\lambda_I}$: $\sigma_i \leftarrow \bot^s$
4 $\sigma_{evic} \leftarrow \bot$
5 **return** $\sigma \leftarrow (\sigma_i)_i, \sigma_{evic}$

qry$^{H_T,H_I}(x, \sigma)$

1 $tag \leftarrow H_T(x)$
2 $i_1 \leftarrow H_I(x)$
3 $i_2 \leftarrow i_1 \oplus H_I(tag)$
4 $a \leftarrow [tag \in \sigma_{i_1} \text{ or } tag \in \sigma_{i_2} \text{ or } tag = \sigma_{evic}]$
5 **return** a

up$^{H_T,H_I}(x, \sigma)$

1 $tag \leftarrow H_T(x)$
2 $i_1 \leftarrow H_I(x)$
3 $i_2 \leftarrow i_1 \oplus H_I(tag)$
4 `// check if up was disabled, first`
5 **if** $\sigma_{evic} \neq \bot$: **return** $\bot, \sigma$
6 `// if tag is already in either bucket`
7 **if** $tag \in \sigma_{i_1}$ or $tag \in \sigma_{i_2}$: **return** $\top, \sigma$
8 `// check if any bucket has empty slots`
9 **for** $i \in \{i_1, i_2\}$ `// in that order`
10   **if** $\mathrm{load}(\sigma_i) < s$
11     $\sigma_i \leftarrow \sigma_i \diamond tag$
12   **return** $\top, \sigma$
13 `// if no empty slots, displace something`
14 $i \leftarrow\!\!\$\ \{i_1, i_2\}$
15 **for** $g \in [num]$
16   $slot \leftarrow\!\!\$\ [s]$
17   $elem \leftarrow \sigma_{i,slot}$ `// element to be evicted`
18   `// swap elem and tag`
19   $\sigma_i[slot] \leftarrow tag;\ tag \leftarrow elem$
20   $i \leftarrow i \oplus H_I(tag)$
21   **if** $\mathrm{load}(\sigma_i) < s$
22     $\sigma_i \leftarrow \sigma_i \diamond tag$
23   **return** $\top, \sigma$
24 `// could not store` x `without an eviction`
25 $\sigma_{evic} \leftarrow tag$ `// last value of` tag `after loop`
26 **return** $\top, \sigma$

Fig. 1: AMQ-PDS syntax instantiation for the Cuckoo filter.

qry$^{H_T,H_I}(x)$	up$^{H_T,H_I}(x)$
1 $tag \Vert i_1 \leftarrow H_T(x)$	1 $tag \Vert i_1 \leftarrow H_T(x)$
2 $i_2 \leftarrow i_1 \oplus H_I(tag)$	2 $i_2 \leftarrow i_1 \oplus H_I(tag)$
3 ...	3 ...

Fig. 2: Changes to apply to Figure 1 to obtain a function-decomposable Cuckoo filter variant. We note that the call to H_I on Line 20 of up^{H_T,H_I} in Figure 1 does not change.

Bibliography

[Blo70] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, jul 1970.

[FAKM14] Bin Fan, Dave G. Andersen, Michael Kaminsky, and Michael D. Mitzenmacher. Cuckoo filter: Practically better than bloom. New York, NY, USA, 2014. Association for Computing Machinery.

[FPUV22] Mia Filic, Kenneth G. Paterson, Anupama Unnikrishnan, and Fernando Virdia. Adversarial correctness and privacy for probabilistic data structures. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 1037–1050. ACM Press, November 2022.