# Enabling Privacy-Aware Zone Exchanges Among Authoritative and Recursive DNS Servers

Nikos Kostopoulos
NETMODE Laboratory
National Technical University of Athens (NTUA), Greece
<nkostopoulos@netmode.ntua.gr>

Dimitris Kalogeras
NETMODE Laboratory
National Technical University of Athens (NTUA), Greece
<dkalo@netmode.ntua.gr>

Vasilis Maglaris
NETMODE Laboratory
National Technical University of Athens (NTUA), Greece
<maglaris@netmode.ntua.gr>

## ABSTRACT

We propose a privacy-aware schema that enables Authoritative DNS Servers to distribute their zones to third parties, e.g. Recursive DNS Servers or scrubbing services, without disclosing sensitive information. Therefore, DNS attack mitigation may be effectively accomplished at external vantage points, presumably closer to the attack sources than the Authoritative DNS Server. Our schema leverages on the space, time and privacy-enhancing properties of Cuckoo Filters to map zone names in an efficient manner, while permitting rapid name updates for large zones. The feasibility of our approach is tested via experiments within our laboratory testbed for a variety of DNS zones. Our evaluation intends to assess the privacy-awareness of our schema and its responsiveness to zone name changes. We conclude that our approach enables mapping of large DNS zones, while preserving privacy.

## CCS CONCEPTS

• **Security and privacy → Network security**.

## KEYWORDS

Domain Name System, Privacy-Aware Zone Exchanges, Distributed DNS Attack Mitigation, Authoritative and Recursive DNS Servers, Cuckoo Filters, Bloom Filters

**ACM Reference Format:**
Nikos Kostopoulos, Dimitris Kalogeras, and Vasilis Maglaris. 2020. Enabling Privacy-Aware Zone Exchanges Among Authoritative and Recursive DNS Servers. In *Applied Networking Research Workshop (ANRW '20), July 27–30, 2020, Online (Meetecho), Spain.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3404868.3406665

## 1 INTRODUCTION

*Recursive DNS Servers* (*Recursors*) are commonly abused by DNS attacks as intermediaries to victim *Authoritative DNS Servers* (*AuthDNS*'s) [1]. In particular, *Water Torture* attacks [2] aim at exhausting computational resources of *AuthDNS*'s by forwarding an immense number of malicious DNS requests through *Recursors.* Typically, these involve randomly generated *Fully Qualified Domain Names* (*FQDN*'s), appropriately crafted to be requested once and thus, bypass the DNS caches of *Recursors.* As a collateral damage, services offered by *Recursors* are degraded since response latency and consumed memory resources increase [3].

Flooding attacks can be mitigated more efficiently close to their origins [4]. However, a complete list of zone *FQDN*'s is commonly not available to *Recursors* as full zone transfers, i.e. *AXFR* type requests are typically restricted by *AuthDNS*'s for security reasons [5, 6]. Thus, effective filtering policies near the attack sources cannot be enforced.

In this paper we propose a privacy-aware schema to facilitate the efficient distribution of *AuthDNS* zones to *Recursors* and/or scrubbing services [7], while being compatible with the existing DNS infrastructure. Our mechanism may benefit *Recursors* that wish to mitigate DNS flooding attacks within their premises, thus safeguarding the *Quality of Service* (*QoS*) offered to their end-users.

We leverage on *Cuckoo Filters* (*CF*'s) [8] to map valid zone names in a hashed format, thus *FQDN*'s are not exposed as plaintext. *CF*'s are time and space efficient probabilistic data structures that enable rapid element lookups in storage and bandwidth constrained applications. *False Positives* (*FP*'s) are possible, resulting in the *AuthDNS* forwarding a regulated volume of *NXDOMAIN* responses, whilst *False Negatives* (*FN*'s) cannot occur for appropriately configured *CF*'s, hence valid DNS requests are never dropped. Contrary to *Bloom Filters* (*BF*'s) [9] that require rebuilding the entire data structure, *CF*'s support dynamic item deletions. Therefore, they are suitable candidates for DNS attack mitigation services that require frequent updates and cannot tolerate any downtime. Our implementation is available in [10].

The paper is structured as follows: *Section* 2 presents background and related work; *Section* 3 provides an overview of our schema; *Section* 4 involves implementation details; *Section* 5 includes the evaluation of our mechanism; *Section* 6 summarizes our work and analyzes future steps.

## 2 BACKGROUND & RELATED WORK

In this section we first describe the functionality and properties of *CF*'s used in our schema to map *AuthDNS* zone names. We then discuss related work that motivated our schema.

### 2.1 Cuckoo Filters (CF's)

*CF*'s [8] are probabilistic data structures that perform time and space efficient set membership tests. The overall memory consumption is decreased since elements are hashed and inserted in the filter, as *fingerprints* in *entries* of a two-dimensional array. Membership tests are always accurate for elements stored in the filter, but may be inaccurate for those not in the filter. Thus, similarly to *BF*'s [9], *FP*'s are possible, whilst *FN*'s are not. However, contrary to *BF*'s, *CF*'s allow for the efficient deletion of previously inserted elements without introducing space overhead and *FN*'s [11, 12].

*CF*'s are characterized by the number of (i) available *buckets m* and (ii) multiple *fingerprint entries b* per *bucket.* An element $x$ is hashed into a *fingerprint* $fgp(x)$ of $f$ bits using the function $fgp()$ [8]. Each element $x$ is assigned to a pair of *buckets* with indices determined by two additional hashing operations. The index of the first *bucket* $h_1(x)$ is determined by hashing the element $x$ using a function $hash()$, which may be based on the same algorithm as $fgp()$. The index of the second *bucket* $h_2(x)$ is determined based on the *Partial-key Cuckoo Hashing* technique [8]; this involves an *XOR* operation between $h_1(x)$ and $fgp(x)$ hashed with the function $hash()$:

$$\begin{cases} h_1(x) = hash(x) \\ h_2(x) = h_1(x) \oplus hash(fgp(x)) \end{cases}$$

The computed *fingerprint* is inserted in either $h_1(x)$ or $h_2(x)$ *bucket* if space is available. Otherwise, an already inserted *fingerprint* is randomly selected from one of these *buckets* and evicted to its alternate *bucket* if an *entry* is available. This swapping process continues until empty space is found or a maximum number of allowed evictions is exceeded resulting in insertion failure. Lookups and deletions involve inspecting if an element's *fingerprint* is in either of its corresponding *buckets*. The *fingerprint* size $f$ [8] for an estimated *FP* ratio $\epsilon$ and maximum *entries* per *bucket b* is:

$$f \geq \lceil log_2(1/\epsilon) + log_2(2b) \rceil$$

The complexity of *CF* element insertion is amortized *O(1)*, whilst that of membership testing and deletions is constant, *O(1)*. Apart from enabling element deletion, *CF*'s outperform *BF*'s in terms of lookup time and memory requirements for applications with *FP* ratio less than 3% and nearly full *CF*'s. *BF*'s remain superior in applications that require frequent insertions of large element sets due to the costly *CF* eviction process. However, our use case does not require frequent bulk insertions, thus we opt for *CF*'s due to their low lookup latency and rapid updates for large zones (see *Section* 4).

### 2.2 Related Work

Various approaches reported in the literature suggest probabilistic data structures (*BF*'s) to enhance DNS performance and security. An *IETF* draft [13] proposed using *BF*'s to map *DNSSEC* zone names in a space efficient, privacy-preserving format for accelerating authenticated responses to requests about invalid *FQDN*'s. However, this proposal may require tools external to DNS [14], i.e. a separate web server that contained *BF*'s, and may not directly support incremental updates as item deletions are not possible in standard *BF*'s.

In [15], it is recommended that *Recursors* use *BF*'s for monitoring DNS requests, thus relying on privacy-aware summaries of sensitive DNS data. The space efficient and hashed nature of *BF*'s enables logging information over long time periods, whilst end-user privacy is not compromised. Thus, *GDPR* [16] directives are not violated. *Recursor* Administrators are required to issue targeted requests over their collected *BF*'s to determine if newly blacklisted domains have occurred within their network, triggering related actions.

Hosts controlled by a particular botnet are based on the same *Domain Generation Algorithm* (*DGA*) [17] to contact their *Command and Control* (*C&C*) servers. Thus, they typically request the same invalid *FQDN*'s. Based on this, [18] maps invalid names requested within a network to *BF*'s on a per host basis. *BF* bits from diverse hosts are compared to detect similar behavior and subsequently identify abused hosts without logging any personal data. In addition, *BF*'s are used to map valid names and efficiently locate *C&C* servers.

Malicious activities related to DNS are usually based on randomly generated names via *DGA*'s. To remedy them in operational environments, some *Recursor* software implementations consider incorporating *BF* functionality. Notably, *PowerDNS* [19] uses *BF*'s to track newly observed names and detect those related to *DGA*'s. *FQDN*'s appearing for the first time are further inspected, whilst reappearing names are considered for resolution. Similarly, *Unbound* [20] includes a learning mode that collects valid *FQDN*'s from *NOERROR* responses and stores them in *BF*'s. When a predefined threshold is reached, the filtering mode is activated and unknown names are dropped. Our proposed schema could significantly increase the accuracy of these approaches by distributing valid *AuthDNS* zone names, while preserving privacy.

Approaches related to DNS *Water Torture* mitigation employ either machine learning [3, 21, 22] or sketch based methods [23, 24]. In previous work [24], we relied on *BF*'s to mitigate *Water Torture* attacks and outlined their privacy-preserving properties for outsourcing *AuthDNS* zone protection to third parties, e.g. *Recursors* and cloud scrubbing services. In this paper, we extend [24] by implementing a privacy-aware mechanism that enables *AuthDNS*'s to communicate their zones in a hashed format, thus distributing mitigation services. Contrary to [24] and related approaches [13, 15, 18] we use *CF*'s, instead of *BF*'s, to reduce our privacy-aware DNS zone sizes and support incremental updates.

## 3 BASELINE DESIGN

This section presents an overview of our mechanism.

### 3.1 Design Requirements

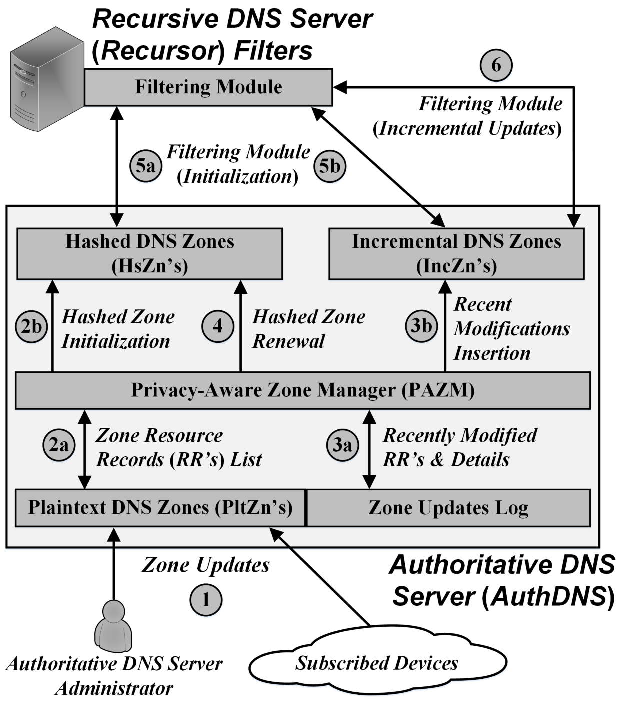
The main design requirements of our schema are:

- *Privacy-aware distribution of AuthDNS zone FQDN's*: The desired system should map the *AuthDNS* zone names not in their actual form, but hashed. This will enable *Recursors* or scrubbing services to retrieve a complete list of all valid *FQDN*'s without inferring the exact zone contents. Thus fine-grained filtering policies may be enforced closer to the DNS flooding attack origins (e.g. bots external to the *AuthDNS* network).
- *Efficient AuthDNS zone mapping*: Our schema should exhibit: (i) compact storage of hashed *FQDN*'s within *AuthDNS*'s, (ii) low latency filtering of malicious DNS requests in *Recursors* and (iii) bandwidth conservation during information transmission among *AuthDNS*'s and *Recursors*.
- *Compatibility with the existing DNS infrastructure*: We require a suitable data serialization format to map hashed *FQDN*'s within *AuthDNS*'s with minor software modifications. Copies of the privacy-aware zones may be obtained using widely adopted types of DNS requests, e.g. *AXFR* [25] and *IXFR* [26].
- *Support for incremental updates*: *Recursors* should be able to update their filtering policies incrementally without downloading again the entire information included in the privacy-aware DNS zones. Thus, the desired system should support flexible name updates.

### 3.2 High-Level Description

Complying with the aforementioned requirements, *Fig.* 1 provides a baseline description of our schema. *Plaintext DNS Zones* (*PltZn*'s) contain the *Resource Records* (*RR*'s) that are under the *AuthDNS* management responsibility. These zones may receive manual and/or *Dynamic DNS* updates [27] either by the *AuthDNS* Administrator or *Subscribed Devices*, e.g. a *DHCP Server*. Details related to the modified *RR*'s are subsequently recorded in the *Zone Updates Log*.

Our system relies on the *Privacy-Aware Zone Manager* (*PAZM*), responsible for constructing and maintaining the privacy-preserving versions of the *PltZn*'s. The *PAZM* recovers a list of the entire plaintext *RR*'s and hashes their corresponding *FQDN*'s to create the *Hashed DNS Zones* (*HsZn*'s). Moreover, recently modified *RR*'s along with details pertaining to them are retrieved from a *Zone Updates Log*. Sensitive information (i.e. names) is hashed, enriched with metadata and included in an *Incremental DNS Zone* (*IncZn*) that reflects recent zone changes. The contents of the *HsZn*'s are renewed in frequent time intervals to match *PltZn* current contents.



Recursive DNS Server (Recursor) Filters
Filtering Module
6
Filtering Module (Incremental Updates)
5a
Filtering Module (Initialization)
5b
Hashed DNS Zones (HsZn's)
Incremental DNS Zones (IncZn's)
2b
Hashed Zone Initialization
4
Hashed Zone Renewal
3b
Recent Modifications Insertion
Privacy-Aware Zone Manager (PAZM)
2a
Zone Resource Records (RR's) List
3a
Recently Modified RR's & Details
Plaintext DNS Zones (PltZn's)
Zone Updates Log
Zone Updates
1
Authoritative DNS Server (AuthDNS)
Authoritative DNS Server Administrator
Subscribed Devices

**Figure 1: Baseline Design of our Proposed Schema**

*Recursors* that wish to filter malicious DNS requests within their infrastructure may retrieve the necessary zone names in a privacy-aware format from the *AuthDNS*. This is possible by initially getting a full copy of an *HsZn* along with its recent modifications from the corresponding *IncZn*. At regular time intervals, *Recursors* may use the *IncZn* contents to incrementally update their *Filtering Modules*. Thus, they avoid the time consuming process of collecting again the entire contents of *HsZn*'s.

## 4 IMPLEMENTATION DETAILS

This section involves implementation details regarding our schema. Specifically, we describe the *PAZM* module and the properties of zones introduced in our architecture, i.e. *Hashed DNS Zones* (*HsZn*'s) and *Incremental DNS Zones* (*IncZn*'s).

## 4.1 Privacy-Aware Zone Manager (PAZM)

This module is responsible for building and maintaining the *CF*'s whose *fingerprints* are used to create and revise the privacy-aware DNS zones. The *PAZM* is implemented in *Python* and relies on a *CF* implementation [28] that we customized to manipulate *CF* contents as desired. *CF*'s in [28] employ the *MurmurHash3* (*Mmh3*) algorithm [29] for both $fgp()$ and $hash()$.

Initially, the *PAZM* retrieves the *Plaintext DNS Zone* (*PltZn*) *RR*'s and retains their *FQDN*. These are hashed into *fingerprints* to fill the *entries* of a *CF*, thus creating the *HsZn*. Notably, an *FQDN* may appear multiple times in the zone, hence only its first appearance is hashed and inserted in the *CF*.

After *HsZn* is initialized, the *PAZM* frequently recovers recent *PltZn* changes from the *Zone Updates Log*. The corresponding names are used to update *CF* contents and changes are incorporated in the *IncZn* via *Dynamic DNS* updates. The *PAZM* ignores changes pertaining to: (i) *RR*'s whose value was updated, thus their *FQDN* was not modified and (ii) *RR*'s that share an *FQDN* with others, but differ in *RR* type and/or value. The latter case is handled by associating frequency counters with each distinct *FQDN*. These monitor how many times an *FQDN* is inserted in the *PltZn* and thus, determine names newly inserted or completely removed from the zone. Otherwise, an *FQDN* could have been inserted multiple times in the *CF* resulting into significant memory overhead.

## 4.2 Hashed DNS Zones (HsZn's)

These zones hold the *FQDN*'s of the *PltZn*'s hashed and mapped in *CF*'s. *Recursors* may retrieve full copies of *HsZn*'s by performing *AXFR* type DNS requests. *Listing* 1 depicts the *HsZn* information serialization format. *CF*-related information is encapsulated by the *PAZM* within *RR*'s of *TXT* type. Note that generic DNS parameters not introduced in our paper, e.g. *TTL* value and standard zone *RR*'s (*SOA*, *NS*, etc.) are not presented.

**Listing 1** Hashed DNS Zones Serialization Format

```
1: ; Zone: hszn.tld
2: ; Cuckoo Filter Parameters
3: buckets.hszn.tld      IN    TXT    <m>
4: entries.hszn.tld      IN    TXT    <b>
5: fgp-size.hszn.tld     IN    TXT    <f>
6: fgp-algo.hszn.tld     IN    TXT    <fgp()>
7: hash-algo.hszn.tld    IN    TXT    <hash()>
8: ; Cuckoo Filter Data
9: <n>.hszn.tld          IN    TXT    <RR Data>
```

The first part of *Listing* 1 (lines 2-7) includes details pertaining to the selected *CF* parameters and algorithms that were introduced in *Section* 2.1. Specifically, the zone file involves information regarding the *CF* $m$, $b$ and $f$ values (lines 3-5). Moreover, details about the algorithms $fgp()$ and $hash()$ are also provided (lines 6-7). Each of the aforementioned details is dedicated an *FQDN*. *FQDN* prefixes are reserved words, properly selected to convey the appropriate meaning.

The second part of *Listing* 1 (lines 8-9) includes *CF* data. The straightforward approach to map *CF fingerprints* would be to assign each *CF bucket* to a separate *RR* of type *TXT*. However, such mapping method would have heavily increased the number of *RR*'s, thus leading to oversized zone files: *RR* fields, i.e. *FQDN*'s, *TTL* values and *Type* parameters introduce unnecessary bandwidth overhead during *AXFR* requests (see *Section* 5.4). Instead, our schema reduces the overall *RR* number by mapping the *fingerprints* of multiple *CF buckets* within a single *RR* of type *TXT*.

*PAZM* maps *CF fingerprints* in the *HsZn*'s as hexadecimal numbers. All *fingerprints* are equally sized of $\lceil f/4 \rceil$ hexadecimal digits. Those requiring less than $\lceil f/4 \rceil$ hexadecimal digits are prepended with 0's.

*Fingerprints* are sequentially inserted within *RR*'s until the maximum *TXT*-type *RR* value size, i.e. 255 Bytes for single strings [30], is reached. *Buckets* that are not full, i.e. contain less than $b$ *fingerprints*, do not have explicit boundaries as the number of stored *fingerprints* varies between 0 and $b - 1$. Thus, they are delimited from the next *bucket* with a dot. Conversely, full *buckets* do not require a trailing dot, hence memory consumption is further reduced. If the contents of a *bucket* do not completely fit within an *RR*, they are split and the remaining part is inserted in the next *RR*. *FQDN* prefixes begin from 0 and increase by 1 until all data are inserted.

*Listing* 2 depicts an *HsZn RR* that maps the first 25 *buckets* of a *CF*. Each *CF bucket* can accommodate up to 4 *fingerprints* and *buckets* with less than 4 *fingerprints* are delimited by dots. The *fingerprint* size is 12 bits, hence each requires 3 hexadecimal digits for mapping in the *HsZn*. Overall, 82 *FQDN fingerprints* are included in *Listing* 2, with the first *fingerprint* of each *bucket* underlined for clarity.

**Listing 2** Hashed DNS Zone RR Data Example

```
0.hszn.tld    IN    TXT    "c64.1dd4d1d590bfbf3ddaa20
3f6cb764b2c647a7063faff67fac8811df81c0fbe65f2.a5a.de2
bcd4666b6f10ba60e5cdc824ee3ba1807bd26d08a3.745a2f8
9e.395cbb723310f27e51c28ee3a96ad2e788092d2514513.44
33be06ed3314bc570ce85c921f5a59e07ee8db11.5f766e444e
96504eb01d090cc0d445.3eb."
```

## 4.3 Incremental DNS Zones (IncZn's)

These zones map the name changes of *PltZn*'s. Therefore, *Recursors* may be based on *IncZn*'s to incrementally update their *Filtering Modules* without downloading the corresponding *HsZn* again. This is possible by performing *IXFR* requests since their most recently checked zone serial number.

**Listing 3** Incremental DNS Zones Serialization Format

```
; Zone: inczn.tld
; Zone Parameters
last-serial.inczn.tld      IN     TXT     <serial>
sequence.inczn.tld         IN     TXT     <seq-no>
; Updates
<n>.inczn.tld  IN  TXT  "<fgp> <action> <h1>,<h2>"
```

*Listing* 3 depicts the serialization format of *IncZn*'s. Each zone involves two parameters (lines 2-4): (i) *last-serial* indicates that *PltZn* modifications prior to this value are incorporated in the corresponding *HsZn*, whilst (ii) *sequence* increments each time the *CF* parameters are altered, e.g. when the filter is full and needs to be reshaped. Thus, *last-serial* defines the starting point for *Recursors* to begin retrieving information from an *IncZn* and *sequence* defines if the related *HsZn* is stale and must be downloaded again.

Each zone *RR* (*Listing* 3, line 6) is related to a single name update and includes (i) the *fingerprint* $fgp$ of the hashed *FQDN* as a hexadecimal number, (ii) the action associated with this *fingerprint*, i.e. *add* (addition) or *del* (deletion) and (iii) the pair of *CF buckets* $h_1$ and $h_2$ corresponding to the hashed *FQDN*. Notably, *RR FQDN* prefixes begin from 0 and increase by one to include each update.

## 5 EVALUATION

Our experiments assess the *HsZn* privacy-awareness, the *HsZn* bandwidth consumption during *AXFR* requests and appropriateness of *CF*'s for *HsZn* management. These are ultimately affected by the selected *CF False Positive* (*FP*) ratio. In a nutshell, large *FP* ratios result into (i) smaller probability of exposing *HsZn* contents, (ii) smaller *CF fingerprint* sizes and thus, less bandwidth consumption for *HsZn* transfers and (iii) larger volumes of *NXDOMAIN* responses by *AuthDNS*'s.

In the following, we compare the effectiveness of *CF*'s versus *BF*-based approaches. We configure a targeted *FP* probability of 0.3% for *CF*'s and *BF*'s. As *CF*'s yield their memory advantages over *BF*'s when they are almost full, we experiment with *CF*'s that have 90% of their *entries* occupied; these values lead to *bucket* sizes that may hold up to 4 *fingerprints*, a common selection in *CF*'s [8]. These parameters define 12-bit *fingerprints* (*Section* 2.1) and thus, 3 hexadecimal characters are required to map each *fingerprint* in the *HsZn*. In contrast, *BF*'s require 3 hash functions [9, 11] to provide a fair comparison against the *CF*-based approach.

### 5.1 Testbed Overview

We performed our experiments within our laboratory testbed. We used a *VM* comprised of 2 vCPU's and 16 GB RAM. The *Hypervisor* was a Dell PE R730 with Intel Xeon E5-2620 v3 2.4 GHz. We selected *BIND9* [31] for our *AuthDNS* as it is a common option among DNS Administrators.

### 5.2 Dataset Description

Access to zone files is typically restricted. However, we gathered datasets for experimentation from servers within our campus and publicly available resources. Specifically, we collected the zone files of:

- *.ntua.gr*: We obtained the names of this zone using *AXFR* requests within our *NTUA* campus zone. This zone involves 8,294 distinct *FQDN*'s.
- *.se*: Names of this zone were obtained via an *AXFR* request as its contents are publicly available [32]. This zone includes 1,387,690 distinct *FQDN*'s.
- *.su & .ru*: The contents of *.su* and *.ru* were exposed for a short time period in 2017 due to misconfigurations in the corresponding *AuthDNS*'s [33]. These zones include 109,719 and 5,325,231 *FQDN*'s respectively. Although these data may be old, they are comparable with the current zone sizes as reported by [34].

### 5.3 Hashed DNS Zones Privacy-Awareness

Although *CF*'s store their elements hashed, attackers may still attempt to gain insight into zone contents. This is possible by performing brute force attacks, i.e. looking up all possible character combinations.

In the following, we assess the capabilities of *CF*'s to withstand brute force attacks in the context of DNS. To that end, we fed a *CF* with all the permitted name combinations varying the first *FQDN* label between 3 and 7 characters. These consist of 37 characters: Latin alphabet, decimal digits and hyphen (not valid as an initial *FQDN* character). Note that these combinations include *Internationalized Domain Names* (*IDN*'s) [35] in our dataset zones.

**Table 1: CF Privacy Properties for DNS**

| 1st Label Length (Characters) | *TP*'s (*FQDN*'s) | *FP*'s (*FQDN*'s) | *FP*'s/*TP*'s (Ratio) |
|---|---|---|---|
| 3 | 320 | 57 | 0.18 |
| 4 | 640 | 1,789 | 2.80 |
| 5 | 1,178 | 68,296 | 57.98 |
| 6 | 1,183 | 2,532,293 | 2,140.57 |
| 7 | 1,363 | 93,665,989 | 68,720.46 |

*Table* 1 depicts the results of our assessment for a *CF* mapping our campus *.ntua.gr* zone. Specifically, *Table* 1 includes the number of *True Positives* (*TP*'s), i.e. matches for *FQDN*'s stored in the *CF*, the number of *False Positives* (*FP*'s), i.e. matches for *FQDN*'s not included in the *CF*, and their ratio.

Attackers may guess *FQDN*'s with first label length of 3 and 4 characters resulting into a relatively small number

**Table 2: Hashed DNS Zones: Bandwidth Consumption during AXFR Requests**

| **Indicative Zone (Distinct *FQDN*'s)** | **Information Serialization Format** | | | ***Cuckoo Filters* (Actual Size)** |
|---|---|---|---|---|
| | ***Cuckoo Filter* (Multiple Buckets / *RR*)** | ***Cuckoo Filter* (Single Bucket / *RR*)** | ***Bloom Filter* (Multiple Bytes / *RR*)** | |
| *ntua.gr* (8,294) | 26.77 KB | 63.91 KB | 41.86 KB | 13.51 KB |
| *su* (109,719) | 352.1 KB | 876.1 KB | 553.11 KB | 178.58 KB |
| *se* (1,387,690) | 4.36 MB | 11.21 MB | 6.86 MB | 2.21 MB |
| *ru* (5,325,231) | 16.78 MB | 43.76 MB | 26.34 MB | 8.46 MB |

of *FP*'s. *FQDN*'s with prefixes longer than 5 characters are protected against brute force attacks with high certainty. Such attacks require exponentially longer time as the prefix grows since the total number of required hashing operations is prohibitively large (approximately 100 billion for 7 characters). Longer *FQDN* prefix lengths result into more *FP*'s, thus discovery of plaintext names is next to impossible. We opted for an *FP* ratio of 0.3% which is sufficient to safeguard privacy. Note that *BF*'s of similar targeted *FP* ratio would have resulted in comparable *TP* and *FP* numbers at the expense of significantly more hashing operations.

## 5.4 Hashed DNS Zones Serialization

In the following, we determine the applicability of diverse information serialization formats for mapping *FQDN*'s into *HsZn*'s. To that end, we triggered *AXFR* requests towards our *AuthDNS* for various DNS zone sizes and recorded their memory footprint.

We considered the following serialization formats: (i) a *CF* with multiple *buckets* mapped within each *RR*, (ii) a *CF* with a single *bucket* corresponding to each *RR* and (iii) a *BF* with each *RR* containing multiple Bytes as hexadecimal numbers. *Table* 2 depicts the bandwidth consumed during *AXFR* requests for the zones of *Section* 5.2 and the serialization formats mentioned above. It also depicts the actual *CF* size, i.e. the space required to hold the zone names in memory.

We observe that mapping plaintext names using a *CF* with multiple *buckets* assigned to each *RR* outperforms the other options. Moreover, although the consumed bandwidth is almost twice that of the in memory *CF* size, the overhead is manageable for modern network links. Thus, the selected *FP* ratio of 0.3% may map diverse zone sizes without issues.

## 5.5 Hashed DNS Zones Management

The *PAZM* performs various operations for managing *HsZn*'s. In the following, we compare the latency of indicative actions using both *BF*'s and *CF*'s to map *PltZn*'s. These include: (i) initial creation of the respective data structure in memory by hashing and inserting all the *PltZn FQDN*'s and (ii) updating the data structures by performing 1,000 deletions and 1,000 insertions. Unlike *CF*'s that directly support deletions, *BF*'s need to be rebuilt excluding the removed data. *Fig*. 2 depicts the latency of performing these operations for the *.ru* zone.

We observe that *BF*'s are created significantly faster than *CF*'s due to the element eviction process during insertions. However, *CF*'s rapidly incorporate changes by employing incremental updates, contrary to *BF*'s that need to be rebuilt as deletions are not supported. Thus, *CF*'s clearly outperform *BF*'s in DNS flood mitigation, where rapid updates are required; data structures are created only during the initialization phase, not affecting *PAZM* operations.

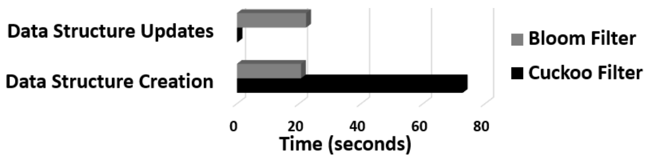


**Figure 2: Latency of HsZn Management Operations**

## 6 CONCLUSION & FUTURE WORK

We leveraged on probabilistic data structures to map the contents of *AuthDNS* zones in a privacy-aware format. This enables effective filtering of DNS flooding within *Recursors* or external scrubbing facilities. We employed *CF*'s due to their time, space and dynamic deletion advantages over *BF*'s. We evaluated our schema via experiments in our laboratory testbed. These indicated that our approach is promising for distributing *AuthDNS* zone names efficiently, while preserving privacy. Thus, mitigation services may be distributed to third parties with no formal collaboration agreements.

As future work, we will investigate recently proposed probabilistic data structures, e.g. *Morton Filters* [36], *Xor Filters* [37] and/or *Vacuum Filters* [38]. These have been reported to outperform *CF*'s in terms of lookup latency and memory requirements, while permitting flexible element updates. Moreover, we plan to investigate potential threats that may arise with our schema and employ data plane programming to protect the open channel used for relaying zone exchanges [39]. We also plan to adapt our schema to the mitigation of amplification *NXNSAttacks* [40] and provide comparison with existing countermeasures [41]. Finally, we will develop a *Distributed* and *Federated Learning* [42] detection mechanism that excludes names infrequently requested by Internet users, thus reducing the size of our zones.

## ACKNOWLEDGMENTS

This work was partially supported by the European Commission H2020 Framework Programme (Grant Agreement No. 856726, GN4-3), the Special Account for Research Funding of NTUA and the ICCS at NTUA under the NSRF 2016-2020 Project HiMiOFOTS, Grant Agreement No. 5002739

## REFERENCES

[1] Scott Hilton. 2016. Dyn Analysis Summary Of Friday October 21 Attack. Dyn. Retrieved June 19, 2020 from https://dyn.com/blog/dyn-analysis-summary-of-friday-october-21-attack/.

[2] John Wagnon. 2018. Lightboard Lessons: The DNS Water Torture Attack. DevCentral. Retrieved June 19, 2020 from https://devcentral.f5.com/s/articles/lightboard-lessons-the-dns-water-torture-attack-32092.

[3] Takuro Yoshida, Kento Kawakami, Ryotaro Kobayashi, Masahiko Kato, Masayuki Okada, and Hiroyuki Kishimoto. 2017. Detection and Filtering System for DNS Water Torture Attacks Relying Only on Domain Name Information. In *Journal of Information Processing (JIP)*, Volume 25, pp. 854-865, September 2017.

[4] Saman T. Zargar, James Joshi, and David Tipper. 2013. A Survey of Defense Mechanisms Against Distributed Denial of Service (DDoS) Flooding Attacks. In *IEEE Communications Surveys and Tutorials*, Volume 15, Issue 4, pp. 2046-2069, 4th Quarter 2013.

[5] Marcin Skwarek, Maciej Korczynski, Wojciech Mazurczyk, and Andrzej Duda. 2019. Characterizing Vulnerability of DNS AXFR Transfers with Global-Scale Scanning. In *Proceedings of the 2019 IEEE Security and Privacy Workshops (SPW)*, pp. 193-198, San Francisco, CA, USA, May 2019.

[6] Valeriy Shevchenko. 2017. DNS Vulnerability for AXFR Queries. Medium. Retrieved June 19, 2020 from https://medium.com/@valeriyshevchenko/dns-vulnerability-for-axfr-queries-58a51972fc4d.

[7] Akamai. 2018. Mitigating DDoS Attacks in Zero Seconds with Proactive Mitigation Controls. Retrieved June 19, 2020 from https://www.akamai.com/us/en/multimedia/documents/white-paper/proactive-ddos-mitigation-with-prolexic-mitigation-controls-whitepaper.pdf.

[8] Bin Fan, David G. Andersen, Michael Kaminsky, and Michael D. Mitzenmacher. 2014. Cuckoo Filter: Practically Better Than Bloom. In *Proceedings of the 10th ACM International Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, pp. 75-88. Sydney, Australia, December 2014.

[9] Burton H. Bloom. 1970. Space/Time Trade-Offs in Hash Coding with Allowable Errors. In *Communications of the ACM*, Volume 13, Issue 7, pp. 422-426, July 1970.

[10] Nikos Kostopoulos, GitHub Account. Latest Commit 2020. Enabling Privacy-Aware Zone Exchanges Among Authoritative and Recursive DNS Servers. Retrieved June 19, 2020 from https://github.com/nkostopoulos/dnspriv.

[11] Sasu Tarkoma, Christian E. Rothenberg, and Eemil Lagerspetz. 2012. Theory and Practice of Bloom Filters for Distributed Systems. In *IEEE Communications Surveys and Tutorials*, Volume 14, Issue 1, pp. 131-155, 1st Quarter 2012.

[12] Li Fan, Pei Cao, Jussara Almeida, and Andrei Z. Broder. 1998. Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol. In *ACM SIGCOMM Computer Communication Review*, Volume 28, Issue 4, pp. 254-265, October 1998.

[13] Steven M. Bellovin. 2001. Using Bloom Filters for Authenticated Yes/No Answers in the DNS. Internet-Draft draft-bellovin-dnsext-bloomfilt-00, Internet Engineering Task Force (IETF), December 2001. Work in Progress.

[14] dns-dir@ops.ietf.org Mailing List. 2001. Using Bloom Filters with DNSSEC. Retrieved June 19, 2020 from https://psg.com/~randy/lists/dns-dir/msg00434.html.

[15] Roland van Rijswijk-Deij, Gijs Rijnders, Matthijs Bomhoff, and Luca Allodi. 2019. Privacy-Conscious Threat Intelligence Using DNSBLoom. In *Proceedings of the 16th IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pp. 98-106. Washington DC, USA, April 2019.

[16] Intersoft Consulting. 2020. GDPR: General Data Protection Regulation. Retrieved June 19, 2020 from https://gdpr-info.eu/.

[17] Daniel Plohmann, Khaled Yakdan, Michael Klatt, Johannes Bader, and Elmar Gerhards-Padilla. 2016. A Comprehensive Measurement Study of Domain Generating Malware. In *Proceedings of the 25th USENIX Security Symposium (USENIX Security)*, pp.263-278. Austin, TX, USA, August 2016.

[18] Hachem Guerid, Karel Mittig, and Ahmed Serhrouchni. 2013. Privacy-Preserving Domain-Flux Botnet Detection in a Large Scale Network. In *Proceedings of the 5th International Conference on Communication Systems and Networks (COMSNETS)*, pp. 1-9. Bangalore, India, January 2013.

[19] PowerDNS Documentation. 2020. Newly Observed Domain Tracking. Retrieved June 19, 2020 from https://docs.powerdns.com/recursor/nod_udr.html.

[20] Daisuke Higashi, GitHub Account. Latest Commit 2016. Random Subdomain Attack Mitigation Using Bloom Filter for Unbound. Retrieved June 19, 2020 from https://github.com/hdais/unbound-bloomfilter.

[21] Yuya Takeuchi, Takuro Yoshida, Ryotaro Kobayashi, Masahiko Kato, and Hiroyuki Kishimoto. 2016. Detection of the DNS Water Torture Attack by Analyzing Features of the Subdomain Name. In *Journal of Information Processing (JIP)*, Volume 24, Issue 5, pp. 793-801, September 2016.

[22] Liguo Chen, Yuedong Zhang, Qi Zhao, Guanggang Geng, and ZhiWei Yan. 2018. Detection of DNS DDoS Attacks with Random Forest Algorithm on Spark. In *Procedia Computer Science 134*, pp. 310-315, 2018.

[23] Shir Landau Feibish, Yehuda Afek, Anat Bremler-Barr, Edith Cohen, and Michal Shagam. 2017. Mitigating DNS Random Subdomain DDoS Attacks by Distinct Heavy Hitters Sketches. In *5th ACM/IEEE Workshop on Hot Topics in Web Systems and Technologies*, pp. 1-6, San Jose, CA, USA, October 2017.

[24] Nikos Kostopoulos, Adam Pavlidis, Marinos Dimolianis, Dimitris Kalogeras, and Vasilis Maglaris. 2019. A Privacy-Preserving Schema for the Detection and Collaborative Mitigation of DNS Water Torture Attacks in Cloud Infrastructures. In *Proceedings of the 8th IEEE International Conference on Cloud Networking (CloudNet)*, pp. 1-6. Coimbra, Portugal, November 2019.

[25] Edward P. Lewis and Alfred Hoenes. 2010. DNS Zone Transfer Protocol (AXFR), RFC 5936.

[26] Masataka Ohta. 1996. Incremental Zone Transfer in DNS. RFC 1995.

[27] Paul A. Vixie, Susan Thomson, Yakov Rekhter, and Jim Bound. 1997. Dynamic Updates in the Domain Name System (DNS UPDATE). RFC 2136.

[28] Huy Do, GitHub Account. Latest Commit 2019. Scalable Cuckoo Filter. Retrieved June 19, 2020 from https://github.com/huydhn/cuckoo-filter.

[29] MurmurHash. Wikipedia. Retrieved June 19, 2020 from https://en.wikipedia.org/wiki/MurmurHash.

[30] Dynu Systems. 2020. TXT Record. Retrieved June 19, 2020 from https://www.dynu.com/Resources/DNS-Records/TXT-Record.

[31] Internet Systems Consortium. BIND 9 Administrator Reference Manual. Retrieved June 19, 2020 from https://www.bind9.net/bind-9.10.8-manual.pdf.

[32] The Swedish Internet Foundation. Internetstiftelsen Zone Data. Retrieved June 19, 2020 from https://zonedata.iis.se/.

[33] Matthew Bryant, GitHub Account. Latest Commit 2017. Summary and Archives of Leaked Russian TLD DNS Data. Retrieved June 19, 2020 from https://github.com/mandatoryprogrammer/RussiaDNSLeak.

[34] DomainTools, 2020, Domain Count Statistics for TLDs. Retrieved June 19, 2020 from http://research.domaintools.com/statistics/tld-counts/.

[35] Internationalized Domain Names. ICANN. Retrieved June 19, 2020 from https://www.icann.org/resources/pages/idn-2012-02-25-en.

[36] Alex D. Breslow and Nuwan S. Jayasena. 2018. Morton Filters: Faster, Space-Efficient Cuckoo Filters via Biasing, Compression, and Decoupled Logical Sparsity. In *Proceedings of the VLDB Endowment*, Volume 11, Issue 9, pp. 1041-1055, May 2018.

[37] Thomas Mueller Graf and Daniel Lemire. 2020. Xor Filters: Faster and Smaller Than Bloom and Cuckoo Filters. In *ACM Journal of Experimental Algorithmics (JEA)*, Volume 25, Issue 1, pp. 1-16, March 2020.

[38] Minmei Wang, Mingxun Zhou, Shouqian Shi, and Chen Qian. 2019. Vacuum Filters: More Space-Efficient and Faster Replacement for Bloom and Cuckoo Filters. In *Proceedings of the VLDB Endowment*, Volume 13, Issue 2, pp. 197-210, October 2019.

[39] Nikos Kostopoulos, Dimitris Kalogeras, and Vasilis Maglaris. 2020. Leveraging on the XDP Framework for the Efficient Mitigation of Water Torture Attacks within Authoritative DNS Servers. To appear in the *Proceedings of the 6th IEEE International Conference on Network Softwarization (NetSoft)*, Virtual Conference, June 2020.

[40] Lior Shafir, Yehuda Afek, and Anat Bremler-Barr. 2020. NXNSAttack: Recursive DNS Inefficiencies and Vulnerabilities. In *arXiv preprint arXiv:2005.09107*.

[41] Petr Špaček. 2020. NXNSAttack: Upgrade Resolvers to Stop New Kind of Random Subdomain Attack. RIPE NCC. Retrieved June 19, 2020 from https://labs.ripe.net/Members/petr_spacek/nxnsattack-upgrade-resolvers-to-stop-new-kind-of-random-subdomain-attack.

[42] Jakub Konečný, H. Brendan McMahan, Felix X. Yu, Peter Richtárik, Ananda T. Suresh, and Dave Bacon. 2016. Federated Learning: Strategies for Improving Communication Efficiency. *arXiv*:1610.05492.