Neuralyzer: Flexible Expiration Times for the Revocation of Online Data

Apostolis Zarras
Technical University of Munich
zarras@sec.in.tum.de

Katharina Kohls
Ruhr-University Bochum
katharina.kohls@rub.de

Markus Dürmuth
Ruhr-University Bochum
markus.duermuth@rub.de

Christina Pöpper
Ruhr-University Bochum & NYU Abu Dhabi
christina.poepper@rub.de

ABSTRACT

Once data is released to the Internet, there is little hope to successfully delete it, as it may have been duplicated, reposted, and archived in multiple places. This poses a significant threat to users' privacy and their right to permanently erase their very own data. One approach to control the implications on privacy is to assign a lifetime value to the published data and ensure that the data is no longer accessible after this point in time. However, such an approach suffers from the inability to successfully predict the right time when the data should vanish. Consequently, the author of the data can only *estimate* the correct time, which unfortunately can cause the premature or belated deletion of data.

This paper tackles the problem of prefixed lifetimes in data deletion from a different angle and argues that alternative approaches are a desideratum for research. In our approach, we consider different criteria when data should be deleted, such as keeping data available as long as there is sufficient interest for it or untimely delete it in cases of excessive accesses. To assist the self-destruction of data, we propose a protocol and develop a prototype, called *Neuralyzer*, which leverages the caching mechanisms of the Domain Name System (DNS) to ensure the successful deletion of data. Our experimental results demonstrate that our approach can completely delete published data while at the same time achieving flexible expiration times varying from few days to several months depending on the users' interest.

1. INTRODUCTION

Social media and cloud storage services have changed the information culture of our society. In the era of Web 2.0 people willingly leave lasting digital traces of their lives while decisions on uploading such information are short-termed. In contrast to analog information, these traces remain available as long as the providers of these services decide to. As a consequence, data such as uploaded documents, communication contents, personal profiles, and posts can be accessed even years after their initial relevance ceased [5, 15, 21, 31]. While the decision to upload personal information to the Internet can be made by each user individually, the control of published data is passed to the service provider. Users depend on a responsible privacy policy while the transparency of the storage and provision process is lost in most cases. In addition, the confidence in the corresponding services has been damaged by a number of data scandals and insights into their archiving practices [16, 19, 22, 23, 32]. Unfortunately, these tendencies conflict with the users' right to be forgotten [8, 12].

As there is no solution on retroactively regaining control over externally stored data, a possible remedy to this problem is a proactive user-driven access control. For instance, the timed revocation of data equips users with control over personal information by revoking the access to data at a specific time after its publication, even if files are maintained by external service providers. Solutions such as Ephemerizer [28], Vanish [14], and EphPub [7] allow users to define a prefixed time when the data will be deleted. All these solutions rely on the very same concept: they encrypt the data and prevent access after the predetermined expiration date by destroying the decryption key. In these solutions, the decryption key is often spread within an existing infrastructure, for example, on distributed hash tables. As long as the key bits can be accessed, the published data remains available. The security goal of all timed revocation schemes is *retrospective privacy*, i. e., the schemes guarantee the revocation of access rights after the expiration time.

However, all the previously-mentioned schemes suffer from the same limitation: users should have *prior knowledge of the correct time* when to delete their shared data. Unfortunately, this is not always feasible and users' privacy preferences are also likely to change over time [1]. For instance, if a user uploads a picture from a party, she may want this picture to be accessible only for a certain period of time. In reality, knowing beforehand when the picture should be deleted is a complicated task that can cause additional overhead to individuals, while it remains most of the times without the desired results; the picture may be destroyed before all of her friends have seen it or it remains available for so long that it may have negative impact.

We address this problem by introducing the concept of *flexible expiration times*. In essence, this concept builds upon the security features of retrospective privacy. Although at first glance it is based on the same infrastructures as the above mentioned systems, at the same time it does not require to predefine an expiration date and therefore removes the respective load from users. Instead, deletion will be based on a suitable revocation model, for example, expiration after interest in the data drops or untimely revocation following excessive access. No matter which revocation model a user selects, the shared data will disappear after a period of time without the user's requirement of selecting this time.

CODASPY'16, March 09-11, 2016, New Orleans, LA, USA
© 2016 ACM. ISBN 978-1-4503-3935-3/16/03...$15.00
DOI: http://dx.doi.org/10.1145/2857705.2857714

Although the concept of flexible expiration times sounds simple enough, its actual design and implementation is not straightforward. The main reason is that we cannot directly modify the architecture of the distributed infrastructures we use. In essence, we require a widely accessible distributed infrastructure where we can design a solution that allows us to react on external systems' behavior without the necessity to modify or revise its architecture. Thus, it is crucial that the ephemeral storage we select provides mechanisms for extending the lifetime of the information we store based on different events (e. g., the receivers' accesses) or at least allow us to perform a roundabout solution to this direction. This constitutes a key difference to the previously mentioned proposals in this space.

In this paper, we investigate this problem space and propose *Neuralyzer*, a timed revocation scheme that allows dynamic access control of users' publicly available data. *Neuralyzer* extends existing approaches by applying flexible expiration times while providing retrospective privacy. More specifically, our prototype is based on the caching mechanisms of the Domain Name System (DNS), similar to the EphPub system [7]. To this end, it uses encryption to protect the data and then splits and distributes the parts of the decryption key over various DNS entries. The key is accessible to anyone who knows which entries has been used for the encoding of the key bits. At the same time, data access leads to the automatic extension of the lifetime of the key bits in the cache of the DNS servers. In essence, the key is valid as long as it is stored in the cache and vanishes once the cache entries are empty. To assess our results, we evaluate the performance of the designed framework regarding data lifetime for different access scenarios (i. e., drop of interest, excessive access, and manual revocation). Based on the results of a simulation study as well as a prototype implementation we show that our approach provides dynamic access revocation to published data. Overall, we believe that *Neuralyzer* can be an important building block to protect users from the long-term exposure of their online data.

In summary, we make the following main contributions:

- We identify the limitations of current schemes for the timed revocation of data and introduce the concept of *flexible expiration times* for online data.
- We propose a protocol to revoke the public access to data that should be forgotten based on three different access heuristics: (i) drop of interest, (ii) excessive access, and (iii) manual revocation.
- We assess the feasibility of our approach by implementing and evaluating a working prototype. Our experimental results demonstrate that our prototype is able to successfully destroy data with flexible expiration times.

2. DESIGN GOALS

In this section, we first state the problem we are addressing and introduce the term of retrospective privacy. Furthermore, we describe three different access heuristics as motivation for the concept of flexible expiration times. Finally, we present the threat model used throughout this paper.

2.1 Problem Statement

The security goal of our approach is to prevent access to shared data after its expiration time, summarized with the term of *retrospective privacy*. This is achieved under the application of different access heuristics which time the revocation of an object.

Retrospective Privacy. With the publication of information in the Internet, all physical control of data is passed to the respective service provider. Timed revocation schemes encrypt valuable information and revoke the access to a respective encryption key once an object should become inaccessible. If access to an expired object is successfully prevented, then *retrospective privacy* is fulfilled.

Access Heuristics. The data should be accessible only for a limited period of time. Hence, the proposed protocol must revoke access rights after that time. Predefined expiration times are the only revocation technique that have so far found attention with respect to the proposal of technical solutions [7, 14, 28]. These approaches are, however, independent of the access heuristic. We argue that predefined expiration times have drawbacks in terms of appropriateness and user friendliness (the users may not know the expiration time nor may want to decide on it beforehand) and thus more dynamic revocations schemes are desirable. Dynamic revocation can be achieved for instance based on the following types of heuristics that take into account the number of accesses over time:
Drop of interest: With reducing audience also the relevance of information can be assumed to disappear. Therefore, the system should detect drops in interest and revoke the accessibility of information to protect its future privacy. In essence, uploads of personal information may be of short-term interest, as such posts are frequently updated and often relate to recent events: A user uploads a picture from an event recently attended, however, does not want to be accessible forever, but only for a period where interest in the event is still present. Through applying the above heuristic, the picture will remain alive by enduring requests, though once the interest drops it will become inaccessible.
Excessive access: Users should be able to revoke the access to publicly available data in case of excessive access. Therefore, a protocol should have the capacity to automatically revoke the access in case of high demand. For instance, an advertising campaign provides free vouchers that should be limited in number. By applying the excessive access heuristic a maximum number of accesses to shared data can be constrained after which the data ceases to exist.
Manual revocation: Manual revocation of objects is an essential fallback method if an applied heuristic does not cover proper deletion. With the capability of manually revoking data any applied fixed or dynamic lifetime of an object can be expired on demand.

We claim that the described, possibly incomplete, list of access heuristics contains instances of desirable behavior. While technical approaches for all heuristics and evaluations of their applicability are desirable, in this paper we chose to focus on the first one (drop of interest) and its technical realization. We later also extend our investigations to the other access heuristics where applicable in the description of our solution and evaluation. We note that providing technical means to address all desirable access heuristics in parallel may not always be possible.

2.2 Threat Model

The security goal of our approach is to provide retrospective privacy, where the adversary is prevented from accessing content after its expiration. We assume that the attacker has no interest in accessing the published data *prior* to its expiration. In other words, the attacker learns the importance of the previously-published data only after that data has expired. This is due to the fact that data is publicly available in the Internet throughout its lifetime and thus is not assumed to be private. The privacy of data is protected only *after* its expiration. We also assume that the attacker can access meta information prior to their destruction, i. e., she can retrieve information about expired or soon to expire data from messages occurring

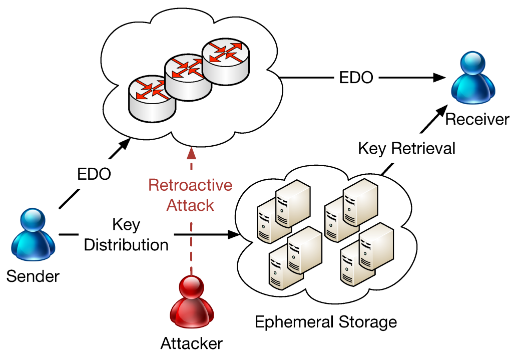


Figure 1: System model.

in the used protocol. Yet, she is not proactively trying to collect information to use it in the future for recovering expired data—if many people utilize the proposed scheme, the sheer amount of information to be collected for later use would simply be excessive.

Within these restrictions, the attacker is able to add custom messages to system (e. g., by injecting new messages, altering existing or replaying previous messages), however, she does not constantly monitor the communication channel of the sender and the receiver. Thus, she is capable of manipulating messages throughout the lifetime of an object, but does not do this in a targeted manner due to a lack of knowledge which objects may be of interest in the future. In addition to the discontinuous monitoring and altering of system-related communication, the attacker may have access to the internal memory of sender and receiver *after* the lifetime of an object.

3. HIGH-LEVEL IDEA

In our proposed model, we define three crucial parts that are involved in the definition of our approach: (i) the ephemeral storage, (ii) the sender, and (iii) the receiver (see Figure 1).

Ephemeral Storage. The lifeblood of our system is the mechanism to store the decryption key. The key should remain accessible only for a valid period of time and then disappear without leaving traces that can be backtracked to its successful recovery. To this end, we define *ephemeral storage* as a mechanism which assures that data is accessible only for a valid period and then disappears permanently. In essence, the ephemeral storage contains the critical information for the successful decryption of data and therefore should always be available and reachable during the data lifetime. This means that it must not suffer from severe down-times and be accessible by the majority of the online population. Therefore, we believe that only a popular infrastructure can serve as ephemeral storage.

Sender. The sender must be aware that, as long the ephemeral key exists, everyone with access to it can decrypt the data. The uploaded data to a server that the user does not own (e. g., Google Drive, Dropbox, etc.), allows to any receiver as well as the server itself to retrieve, decode, and store it permanently. Previous works made it harder for the server to accurately use the collected data [10, 26], but these approaches are outside the scope of this paper. Following our threat model, we assume that the server does not proactively collect this data.[1] With that in mind, the sender compiles the data to a data structure called Ephemeral Data Object (EDO) which contains the encrypted data and a link to the ephemeral storage that includes information for the construction of the decryption key, and then uploads it to a server from where it can be retrieved.

[1]If this assumption is not fulfilled, out-of-band channels can be set up and used to distribute the information required for key recovery.

Table 1: Summary of notation.

Notation	Explanation
S	: Sender
R	: Receiver
N	: Length of DNS portrayal
I	: Length of decryption key
$\|K\|_I : k_i \in K$	: Key of length I
$\|K'\|_I : k_i \in K'$	: Recovered key of length I
$\|C\|_{I\times N} : c_{i,n} \in C$	: Cache entries for key
$\|D\|_{I\times N} : d_{i,n} \in D$	: Domains of cache entries
$\|TTL\|_{I\times N} : ttl_{i,n} \in TTL$	: TTL values of cache entries
x	: Key threshold
t_1, t_2	: TTL thresholds

Receiver. A receiver who retrieves an EDO is able to decode it and decrypt its data before the key expires. To do so, the key must first be recovered from the ephemeral storage and then be reconstructed locally. Additionally, in our proposed model and following the *drop of interest* access heuristic, the receiver contributes to the viability of the EDO. Thus, it is crucial that the ephemeral storage provides mechanisms for extending the lifetime of the stored key based on the receivers' access. This way, the data will be accessible as long as there is sufficient interest for it and disappear afterward.

4. SCHEME DESCRIPTION

In this section, we provide the design details of our approach. We propose the utilization of the DNS resolvers' caching mechanisms as an instantiation for the ephemeral storage. In the following text, we first introduce the concept of *ephemeral bits* and then describe the details of the protocol design.

4.1 Ephemeral Bits

As in any other encryption system, data is encrypted to protect it from unauthorized access. Given that the encryption algorithm remains the same, the size of the key defines its resistance to brute force attacks. For instance, breaking a symmetric 256-bit key, which is used to encrypt data with the AES algorithm [9], by brute force requires 2^{128} times more computational power than a 128-bit key. In our approach, we use the same principle of defining the size of a key by dividing it into its key bits. However, and given the fact that we require the bits to be available only for a certain period of time and then automatically being destroyed, we refer to these key bits as *ephemeral bits*.

In our proposed scheme, we use the caching mechanisms of DNS resolvers to encode the ephemeral bits and enhance them with the property of self destruction. The utilization of DNS resolvers is well-suited for our approach because of their automatic mechanism to clear their caches within a predefined *Time to Live* (TTL) period [24]. Consequently, after a scheduled timeout, the key will disappear from DNS caches and the data can no longer be decrypted.

4.2 Protocol Description

Our approach allows data to be vanished once the interest for it drops. This way, if a retroactive attacker attempts to access the data after that point, she cannot recover it anymore. To do so, we introduce the data structure EDO that protects the publicly accessible data by encrypting it, encapsulating the encrypted content, and ensuring its disappearance when the expiration time is reached. The data contained in EDO becomes useless after a period of time, even if an attacker retroactively obtains a valid copy of the EDO. Overall, the lifetime of an EDO is divided into three phases: (i) construction, (ii) access, and (iii) revocation. For convenience Table 1 summarizes the notation we use throughout this section.

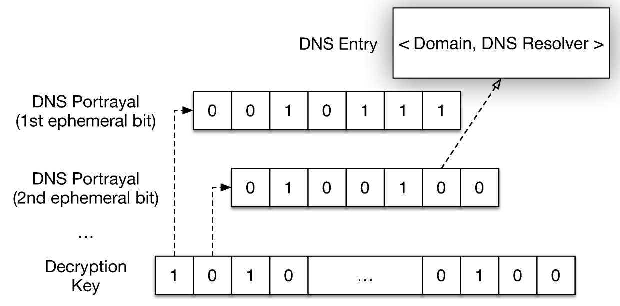


Figure 2: An example of key distribution in DNS servers. We note that initially, all entries in the DNS portrayal of a key bit `0` are in fact `0` (some of them may get set over time, as shown in the figure).

4.2.1 Construction Phase

In this phase, a sender S uploads to a third-party server data that will cease to exist after the interest for it drops. For each new EDO, our algorithm executes the following steps:

Step C1. First, it generates a random key. The size of the key defines the security of the encrypted data and therefore we recommend at least a 128-bit key. Then, it utilizes this key to encrypt the data using the AES algorithm.

Step C2. Next, it converts the key bits into ephemeral bits. Specifically, each ephemeral bit is encoded to a list of DNS entries, which we call *DNS portrayal*. Each DNS entry is represented by a pair of a domain name and a DNS resolver, illustrated in Figure 2. For this purpose, we use a list of DNS resolvers and a list of precompiled domains. The latter can be generated either by crawling the web or by selecting random IP addresses and performing reverse DNS lookups (details on how this can be achieved practically are provided in Section 4.3.1). Note that the selected domains must not be preloaded in the cache of the associated DNS resolvers. With that in mind, our algorithm can assign the value `1` in a DNS entry by performing a *recursive* DNS request which will set the cache entry.

For all bits k_i of the key, we proceed as follows: We do not take any action if $k_i = \texttt{0}$ (apart of a non-recursive DNS query verifying that the respective DNS entry is indeed not set). For every $k_i = \texttt{1}$, we execute recursive DNS queries for the respective domain names (defined in the EDO). As a result, values of the DNS portrayal for each key bit `1` are set to active (loaded in the cache), while the DNS portrayal for each key bit `0` remains unset (respective domain names are not in the cache).

Note that we do not initially assign the value `1` to *all* entries in the DNS portrayal if $k_i = \texttt{1}$, but only to a certain number of them. Actually, we want to avoid that all DNS resolvers happen to clear their caches at the same time, which would prevent key updates during the access phase (detailed further below). The length of the DNS portrayal (N) and the number of entries required for a successful representation of an ephemeral bit (x) are defined by thresholds. We discuss the threshold values in Sections 4.3 and 5.

Step C3. In this final step, our algorithm compiles the EDO. An EDO contains the encrypted content and the list of domains D for each key bit k_i, represented by N cache entries $c_{i,n}$. As we will present in the access phase, this information is sufficient for successfully decrypting the EDO during its lifetime.

4.2.2 Access Phase

Once the EDO has been compiled, it can be distributed to third-party servers. A receiver R can retrieve the EDO and access its content as follows:

Step A1. First, it retrieves the encrypted content and extracts the list of domains D for each k_i from the EDO.

Step A2. In order to assign the correct values to each ephemeral key bit, our algorithm performs *non-recursive* queries to the DNS resolvers for their corresponding domain names. If the resolver contains the domain in its cache, our algorithm assigns the value `1` to this DNS entry otherwise the value `0`. To minimize the errors that may occur from externally modified entries, we use an empirically calculated threshold x. If the sum of the returned values exceed this threshold, the algorithm sets the corresponding ephemeral key bit:

$$\texttt{recover}\begin{cases}\sum_{n=0}^{N-1} c_{i,n} \geq x : 1 \\ \sum_{n=0}^{N-1} c_{i,n} < x : 0\end{cases}, \tag{1}$$

where x is the threshold value that enables for recovering errors in the cache entries. Such errors may be induced by random recursive DNS requests leading to $0 \rightarrow 1$ switches (due to DNS queries from users external to our scheme during the execution of the protocol), or failures at the DNS cache resulting in $1 \rightarrow 0$ switches (the DNS server has emptied its cache, for example, due to a reboot).

Step A3. In this step our algorithm extends the lifetime of EDO. To do so, it updates a random DNS entry of each DNS portrayal that represents an ephemeral key with the value `1` by executing a recursive request. However, we want to have variation in the remaining TTL values on the entries of each portrayal. This minimizes the danger of having resolvers that simultaneously empty their caches or at similar times. We achieve this by performing a recursive DNS request when the median or minimum TTL per DNS-portrayal is less than a preselected threshold. For instance, we can update the DNS entries if we see that the median TTL is less than $\text{TTL}_{max}/2$, or if the DNS entry with the minimum remaining time to empty its cache is less than $\text{TTL}_{max}/10$, where TTL_{max} is the maximum value assigned to the TTL by the DNS servers. It is worth to mention here that both conditions work in parallel and we update a DNS entry whenever one of these conditions, or both of them, is satisfied. In essence, without refreshing the cache entries, an object would expire as soon as a significant amount of 1-bit representations has switched from 1 to 0. Overall, to prolong the initial lifetime limit, each receiver performs a cache refreshment after a successful reconstruction of the EDO for all key bits $k_i = 1$:

$$\texttt{refresh}\begin{cases}(\texttt{median}(ttl_{i,n}) < t_1) \vee (\exists n : ttl_{i,n} < t_2) : 1 \\ (\texttt{median}(ttl_{i,n}) \geq t_1) \wedge (\forall n : ttl_{i,n} \geq t_2) : 0\end{cases}, \tag{2}$$

where the `refresh` operation is only executed in case the median TTL $\texttt{median}(ttl_{i,n})$ or the TTL of a single value $ttl_{i,n}$ for a key bit k_i fall below thresholds t_1 or t_2. In this case, a random $c_{i,n}$ currently 0 is set to 1 by a recursive DNS request to the respective domain $d_{i,n}$. Note that the threshold values as well as the metrics (median/minimum TTL) can be adapted to a specific deployment scenario and are not necessarily bound to the above definition.

Step A4. In the final step of this phase, the ephemeral key has been successfully reconstructed. Then, the receiver uses this key to decrypt and access the encrypted data.

4.2.3 Revocation Phase

This is the last phase in the lifetime of an EDO. We expect that after some time there will be a drop of interest for a published EDO. Consequently, the number of accesses will decrease as well, which will cause *Step A3* of the access phase to be executed less and less frequently. This results in cleared caches of the DNS entries, such that, the ephemeral key cannot be reconstructed successfully. Therefore, the data will vanish, i. e., the encrypted data will be present but will be useless without a retrievable decryption key.

4.2.4 *Different Access Heuristics*

In this section, we discussed how our model operates in the case of *"drop of interest"*. We now describe how our scheme could handle different access heuristics with only small modifications.

Excessive Access. In this case, we want to revoke the access of an EDO in case the interest for it exceeds a certain upper bound of allowed accesses. For this reason we need to count the number of accesses to the EDO. As the DNS resolvers do not have an access counter that would be visible or accessible to normal users, we can enrich the EDO with a probabilistic self-destruction mechanism. More precisely, for every single access of EDO, we generate a random number. The decision to destroy the decryption key is based on whether the generated random number is larger than a specific bound defined by the highest number of acceptable accesses (e. g., for a allowed accesses, the bound would be $1 - 1/a$ if the random numbers are selected from $[0, 1]$). If the result leads to not allowing more accesses, the ephemeral key gets destroyed by performing recursive DNS requests for all key bits. This causes all the ephemeral bits to take the value `1`. In essence, this approach is like a dice with N sides and if we throw the proper side the ephemeral key is destroyed. This requires the sender to include the maximum number of accesses in the EDO and is based on the assumption that receivers of the EDO before the expiration time are not malicious (in accordance to our threat model).

Manual Revocation. Additionally, our approach supports the revocation of an EDO at a time its creator decides to. This can be done by performing recursive DNS requests to all DNS entries (similar to what we previously discussed). After that, the data will not be accessible any more, which is similar for a user to own the server in which the data was initially uploaded to and then decides to remove the data from the server. That said, we are aware that manual revocation entails the danger of a receiver or an attacker prematurely destroying the key. However, as we described in our threat model we try to protect the data only against a retroactive attacker, thus a proactive attacker is outside the scope of this paper.

4.3 Instantiation of the Scheme

Central to the application of the proposed scheme are the list of domain names used in the construction of the EDO and the length and threshold of the DNS portrayal. We detail both in the following. We then also reason about the error correction capabilities and scalability issues of the proposed scheme.

4.3.1 *List of Precompiled Domains*

The domain names which are used in our prototype are collected automatically by utilizing reverse DNS lookup. This method is based on the DNS infrastructure and allows the resolution of an IP address to its designated domain name (also known as forward DNS resolution). To generate a list of domains, we perform reverse DNS lookups to a range of IP addresses. In this procedure, we exclude addresses that have been reserved for special purposes by the Internet Engineering Task Force (IETF) and the Internet Assigned Numbers Authority (IANA). Before employing the domains we ensure using non-recursive DNS lookups that the domains are not currently cached in the DNS resolvers.

An alternative way to generate such a list is by crawling the Internet [30], using heuristics to reach less likely used websites. In both cases, our list of domain names should contain rarely used websites in order to reduce the chance of interference from legitimate DNS lookups. However, the randomized reverse DNS lookup approach above has the advantage that different users are more likely to select different domains so that their key storages do not interfere with each other.

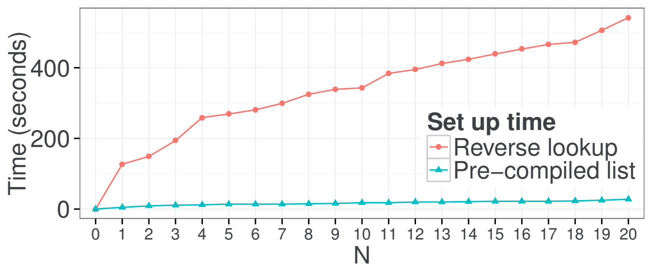


Figure 3: Time to construct an EDO for various lengths of the DNS portrayal.

Figure 3 illustrates an example of EDO generation with and without a precompiled list of domains for different numbers of DNS servers used for each key bit (size N of the DNS portrayal). The time consumption increases with a higher number of lookups. Measurements of 1024 random lookups (as would, e. g., be needed for $N = 8$) indicate an average duration of 0.31 seconds per successful operation. In this setup, no collisions of domains were encountered. We note that the domain lists can be compiled by the sender in advance to creating and publishing an EDO such that this does not create a bottleneck in the execution of the system.

4.3.2 *Size of DNS Portrayal*

The length N and the threshold x of the DNS portrayal determine the number of errors that our scheme can handle and influence how long the lifetime of the data can be extended. We consider as error any unexpected modification of the cache of a DNS resolver. That is, (i) a DNS resolver that empties its cache before the prescheduled TTL or (ii) an "accidental" recursive DNS request to our selected domain. To find representative error numbers, we monitored how the DNS entries behave in the real world. Our measurements of error frequencies are based on a set of 1853 DNS resolvers that had shown reliable service over a period of several months. We monitored the cached and uncached entries of 1000 randomly selected domains. For each resolver we collected all errors that occurred during the entire lifetime of entries. Results show that (i) with an error probability of $e_{1\to 0} = 0.7\%$ a resolver empties its cache before the scheduled TTL expires and (ii) $e_{0\to 1} = 0.1\%$ of accidental recursive requests were performed on the observed set of domains during out measurements. Although the exact error rates may vary over time, these numbers provide reasonable estimates for our derivations and simulation.

Both types of errors, that is bit flips $e_{1\to 0}$ and $e_{0\to 1}$, should be handled by our scheme. Based on this insight we can derive a minimal parameter setup $N = 3$ and $x = 2$ that enables correcting errors of type (i) and (ii) while providing the capability of refreshing cache entries. During initialization, i. e., when the key bits are first stored in the portrayal, the number of bits set to 1 must be larger than x; in later investigations, we use $\lceil \frac{N}{2} \rceil$ 1s for the initialization.

Beyond the minimal setup, an increase of the portrayal length N allows for increasing the scheme's correction capability whereas an increasing x threshold leads to a higher overhead in the presentation of bits (see Equation 5 for reference).

The initial lifetime (before lifetime extensions due to new accesses to the object) are given by the TTLs. Typical TTL values we observed for 1000 random lookups are characterized as follows: median of 86,400 s = 24 h, min. 72 s, max. 604,800 s = 14 days, standard deviation 110,704 s ≈ 30 h. Due to variations in the TTLs as well as due to errors, we initially do not set only one but multiple entries in each portrayal to 1.

4.3.3 Error Correction

Errors can cause the destruction of an encryption key before the heuristic would have triggered the revocation of an object. The application of a correction scheme should compensate such errors as long as the heuristics consider an object accessible. We consider the regular expiration of an entry as type-*(iii)* error for this purpose: as long as the applied heuristics did not trigger the destruction of the key, all changes in the key representation should not lead to the expiration of an object.

As the expiration of an entry is a necessary event, the error probabilities for $e_{1\rightarrow 0}$ and $e_{0\rightarrow 1}$ errors are highly distinct. Previous revocation systems utilized different correction schemes with individual characteristics: EphPub [7] uses Reed-Solomon (RS) error correcting codes (ECC) and Vanish [14] uses Shamir's Secret Sharing (SSS). The following paragraphs compare the performance of RS and SSS with the portrayal scheme applied in our scheme.

Error Correcting Codes. The RS code is an optimal BCH code (in non-binary mode) designed for correcting burst errors and it therefore performs best for errors that occur in row in an encoded word. In context of our scheme, the majority of type-*(iii)* errors occur randomly in 50 % of the cache entries and—since DNS servers and domain names are picked and distributed using a random selection—are uniformly distributed rather than bursty. An alternate to burst-optimal RS codes are Golay codes. The extended binary Golay code $[24, 12, 8]$ can correct uniformly distributed errors and would be more suitable for the occurrences of errors in our scheme. Different to standard ECC schemes, the parameters of our portrayal can be adapted to the type of errors occurring, that is the asymmetric distribution of 0- and 1-errors. Given this characteristic it is possible to achieve high correction rates while applying a smaller overhead to the key representation. With respect to the applied overhead and correction capabilities, the portrayal outperforms standard ECC schemes; we provide a detailed derivation in Appendix A. Thus, the portrayal is selected for providing robust lifetimes and the possibility of refreshing the key representation of an object.

Shamir's Secret Sharing. Secret sharing schemes distribute portions of a secret message over several users where a threshold t defines the number of shares required for reconstructing the original secret. When applied to an encryption key, the key can only be recovered when at least t shares are available at access time. Threshold values close to the number of shares lead to fast revocation and high security while smaller t values lead to a more robust system that can survive a higher number of errors. Overall, secret-sharing can provide theoretical security for revocation schemes that *do not* rely on the refreshing of entries: as soon as a bit error occurs in one share it becomes invalid and cannot be used for reconstructing the key. As our scheme requires the refreshing of bits, it must be robust to bit errors occurring through the expiration of cached entries.

4.3.4 Scalability

Without any centralized component in our proposed scheme it is not possible to store the domains that are already used for existing objects. Therefore it is not transparent whether an uncached entry for a domain is currently unused or represents a 0-entry or erroneous 1-entry in another portrayal. The probability of overlapping with an existing portrayal depends on the number of active domains: for instance for 510 million domain names [18] and a minimal portrayal with $N = 3$ and $x = 2$ overall 1,328,125 parallel users can share data via our model. The probability of overlapping entries at the initialization is approximately $1\,\%$ for 28,000 parallel users, as $50\,\%$ of bits in a portrayal are 0-bits that are prone to overlap with entries that are already used.

Table 2: Parameter space of simulation study.

Parameter	Variable space
I	$: 128$
N	$: [4, 5, ..., 10]$
x	$: \{1, 2\}$
t	$: 43, 200$
$e_{1\rightarrow 0}$	$: [0, 0.007]$
$e_{0\rightarrow 1}$	$: [0, 0.001]$
TTL_{min}	$: 1, 200$
TTL_{max}	$: 604, 800$

We note that high numbers of parallel users can increase the original probability for $e_{0\rightarrow 1}$ errors in addition to the error rate induced by accidentally performed recursive requests. In order to increase the robustness of our scheme for scenarios of massive parallel use, an additional encoding can be applied. For instance, in the portrayal of the encryption key, each bit can be encoded by a mixed tuple. A possible implementation could use a Manchester encoding, where 0-bits are encoded by 01 while 1-bits are encoded by 10. This allows to detect all 0 key bits that are already in use under the expense of doubled overhead.

5. SIMULATION STUDY

We conduct a simulation study to explore the parameter space, with respect to three performance parameters for the access heuristics of dropping interest and excessive access.

5.1 Simulation Setup

The parameters we investigate are shown in Table 2. They include the range of possible portrayal lengths N and the threshold values x and t for key recovery and TTL refreshing for a fixed key length $I = 128$ bit. We simulate the caching of entries and their TTL values, possible errors induced to the key shares, and different types of access patterns representing fluctuating numbers of receiver requests. The possibility of errors in the cache entries $e_{0\rightarrow 1}$ and $e_{1\rightarrow 0}$ each define the probability of a cache entry being flipped and are derived from measurements on the set of resolvers. The assignment of TTL values follows the measurements of EphPub [7].

Within a simulation run the lifetime of an object is monitored starting with its initialization until the destruction of the encryption key. It is executed step-wise with each step representing one second of object lifetime. The number of requests to an object at a time is simulated through different probability distribution functions (PDF): we applied an exponential and shifted normal distribution for the scenarios of dropping interest and an inverse exponential PDF for the scenario of excessive accesses. At each simulation step the cumulative number users performs an access on the object.

5.2 Performance Parameters

We consider the following performance parameters:

- *Sensitivity of expiration*: The sensitivity of expiration refers to the point of time when the access to an object gets revoked. While systems with high sensitivity react to the underlying metrics at an early point in time, more robust parameter setups extend the lifetime of an object.

- *Reliability of results*: For identical parameter setups, the lifetime of an object may vary in multiple simulation runs, as the system dynamics lead to a probabilistic model. Nevertheless, the reliable revocation of an object should still be ensured. We analyze the continuity of object lifetimes for highly sensitive as well as robust parameter setups.

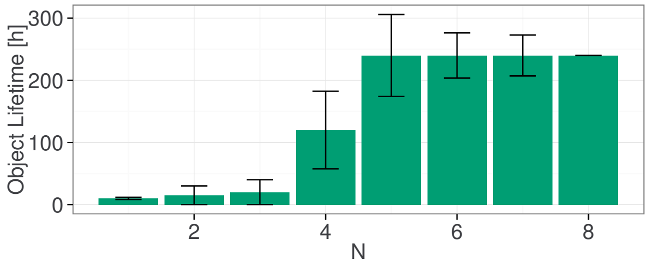


Figure 4: Error correction capabilities for $x = 2$ and increasing N. The error bars summarize the standard deviation of object lifetimes for 50 repetitions.

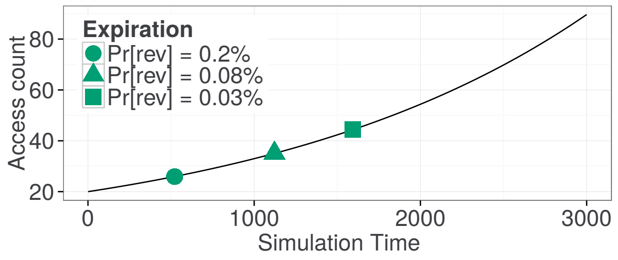


Figure 5: Lifetimes for $x = 2$, $N = 10$ and access heuristic for excessive access at an inverse exponential distribution of accesses.

- *Error correction capabilities*: Based on the optimal threshold $x = 2$, the correction capabilities can be increased by adapting the portrayal size N. Therefore a variation of the portrayal length leads to different object lifetimes.

5.3 Sensitivity of Expiration

Figure 4 summarizes the object lifetime for an increasing portrayal length N at the optimal threshold $x = 2$. With an increasing ratio $\frac{N}{x}$ the correction capabilities increase and lead to a prolonged object lifetime. This characteristic can be applied for defining the sensitivity of metrics in our scheme: to provide a robust revocation mechanism we need a ratio of at least 2.5 while smaller ratios lead to a system sensitive to a lack of refresh operations.

We measured the expiration time of an object with respect to exponentially (Figure 6a) and normally (Figure 6b) distributed access patterns. To provide an object with low sensitivity to dropping access rates, we simulated the object lifetime with $N = 10$, $x = 2$ and therefore a high ratio $\frac{N}{x} = 5$. For a scenario with access rates similar to a normal distribution the object expires as soon as the number of users is close to zero. In a parameter setup with increased sensitivity and $N = 4$, $x = 2$, $\frac{N}{x} = 2$ the expiration is triggered earlier, as a reduced number of expirations in cache entries can be covered. For a scenario with exponentially distributed access rates a highly sensitive parameter setup would require a more restrictive parameter setup.

Other than the Drop of interest heuristic, in case of Excessive access the key information should be destroyed if the number of accesses increases dramatically. Based on a probabilistic approach the expiration of an object can be reduced or prolonged by adapting the probability of destruction. As shown in Figure 5 higher probabilities lead to an early destruction of key information while reduced probabilities prolong the object lifetime. Nevertheless the revocation of an object cannot be determined as reliable as with the Drop of interest heuristic, as the moment of expiration is random.

5.4 Reliability of Results

Given the above parameter setups for a sensitive or robust revocation we analyzed the reliability of results for multiple repetitions. As shown in Figure 6, the dynamic characteristics of the simulation model lead to variations in the object lifetime: for 50 repetitions with TTL values distributed according to the measurements of common lifetimes, the results show slight deviations in the expiration time. Nevertheless the overall range is limited leading to a reliable destruction of key information for a given parameter set. Based on these characteristics it is possible to initialize an object with a target sensibility of the Drop of interest heuristic.

5.5 Error Correction Capabilities

Given the probability of naturally occurring errors in the cache representation of key bits, the x threshold is capable of correcting a limited amount of errors. We tested different x-thresholds based on a cache dimension of $N = 20$, which represents an acceptable creation duration. We applied the best performing setup to a simulation scenario with increasing error rates and accumulated the results for 100 random repetitions. As summarized in Figure 7, an average error probability of 10^{-4} already leads to a major decrease in the object lifetime when no correction threshold is applied (Figure 7a). However, $x = 2$ provides a constant lifetime up to an error rate of 10^{-3} and still allows for a performing system at 10^{-2} (see highlighted area), which is beyond the realistic error rates measured in the prototype implementation.

Overall, the simulation results lead to the conclusion that an application of the Drop of interest heuristic is possible with our scheme. Furthermore, a variation of system parameters such as the x-threshold enable for adjusting the sensitivity of the applied heuristic leading to a shift in the expiration time.

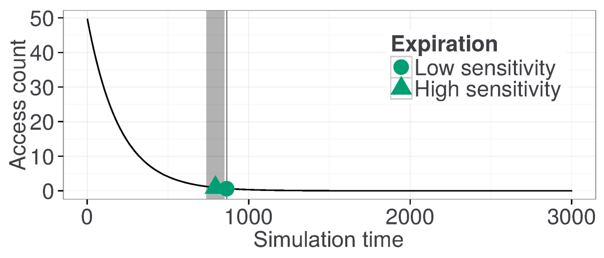
6. PROTOTYPE EVALUATION

To demonstrate the viability of our approach, we have implemented our proposed scheme as an autonomous framework called *Neuralyzer*. Our prototype is capable of dynamically encrypting data with a randomly generated key and then distribute the key bits across multiple real world DNS resolvers. To this end, we utilized the Python programming language and more specifically the PyDNS module, which allowed us to send both recursive and non-recursive DNS queries to our chosen DNS resolvers. Additionally, we used the AES algorithm (with a standard key size of 128 bits) of PyCrypto module to encrypt the data. Finally, each compiled EDO is encoded in a base64 format which is ideal for “shipping” the data across the Internet.

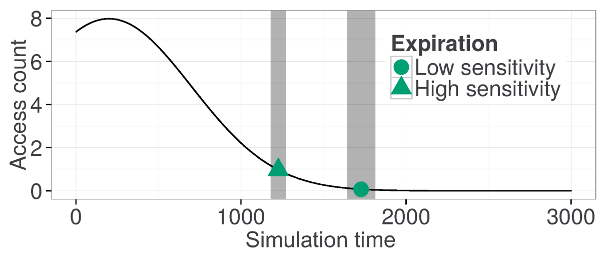
6.1 Expiration Time

We first consider the expiration of the publicly available data once the interest for it ceases to exist. In other words, the decryption keys of EDO should be available only as long as there is sufficient interest and disappear afterward.

To examine whether our protocol design is accurate we created two sets of EDO documents. Each set contained 100 documents compiled with our prototype. On the first set we did not apply any accesses at all, whereas on the second set we applied sufficient accesses (2 to 10 accesses per hour, random selection of their number and point in time) for a day, which is a typical lifespan for a normal tweet [33]. To assess the viability of the generated key bits in the DNS resolvers we performed non-recursive DNS queries to the corresponding servers. This way we did not affect the caches of the resolvers while we were able to monitor the TTL values on their caches, which means that we could monitor the existence of the decryption key without interfering with our experiment.

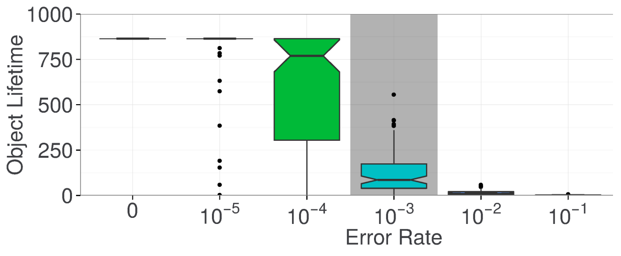


(a) Exponential Distribution.

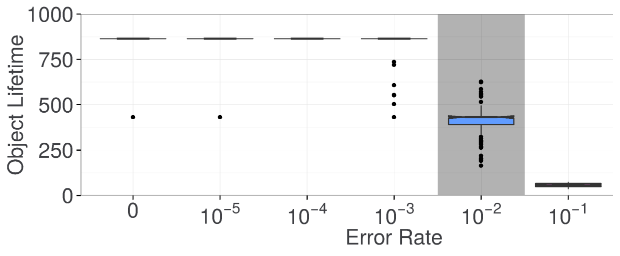


(b) Shifted Normal Distribution.

Figure 6: Expiration time for 50 repetitions of sensitive and robust parameter setup. The highlighted boxes represent the positive and negative standard deviation of measured lifetimes.



(a) Accumulated lifetime of EDO for $x = 0$ (no error correction).



(b) Accumulated lifetime of EDO for $x = 2$ (minimal correction threshold).

Figure 7: EDO lifetime measured for different error rates at $N = 20$.

The outcomes revealed that in both scenarios the keys were destroyed during the expected time (same behavior for all documents). More precisely, the keys were vanished once the cache of the corresponding DNS resolvers emptied. Keep in mind that the TTL of the DNS resolvers cache define for how long the data will be available after their last access. In the first scenario, we noticed that after the first hour we could not decrypt the EDO documents any more. This happened due to the fact that the majority of the resolvers provided us with a TTL of 3600 seconds. In the second scenario, the keys remained available for the time that there existed sufficient interest for the documents. The major take away message from this experiment is the permanent deletion of the key once the interest drops. One anecdotal experience of this experiment is that we noticed during the day few resolvers to prematurely empty their caches or recursive DNS requests. However, none of these incidents were sufficient to destroy the decryption key.

6.2 Long Lasting Data

In contrast to certain types of data such as status updates on social media, comments on news websites, or publicly available pictures in cloud services in which the expected interest will be very limited and ranging from some hours to few days, there exist others such as posts on popular blogs or project documents that their creators expect to last longer, e. g., weeks or even months. The design of our model is meant to maintain the data as long as there is sufficient interest for it. We thus wanted to see if our approach operates in the real world as it is designed to, or an increased number of random recursive queries and errors in the caches of DNS resolvers over time will prematurely destroy the access to the compiled data.

To examine how *Neuralyzer* behaves in such long lasting data we created EDO documents, which we then accessed for a period of 33 days. In essence, we wanted to measure how a drop of interest scenario will behave in long lasting objects (in our case for a period of one month). Thus, the number of accesses dropped from over 400 accesses per day, which was at the beginning, to only a few accesses per day at the end. The outcomes of our experiments demonstrated that the number of random recursive queries and errors in the caches of DNS resolvers were not sufficient to revoke the access to the compiled EDO. More precisely, we were able to successfully reconstruct the decryption keys for a period of 30 days, as long as there was sufficient interest for the published data, and after that point in time the data were not accessible any more as Figure 8 illustrates. After the 33rd day there was no more accesses and this is the reason for the linear drop of accesses after this point in time. Consequently, this experiment along with the previous one demonstrates that *Neuralyzer* can achieve expiration times varying from hours to several weeks or months, depending on users' interest.

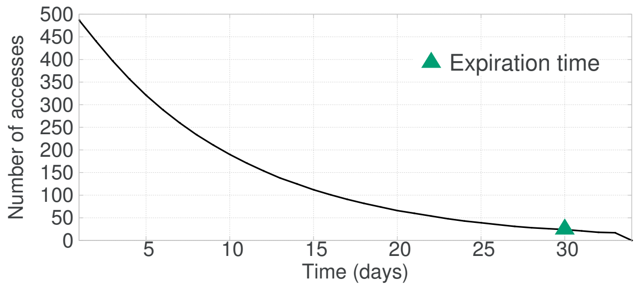


Figure 8: Number of accesses for a period of one month.

6.3 Accuracy of Excessive Accesses

Most of the times users do not really care about the publicity of their exposed information, e. g., status updates or tweets. However, there exist cases that the number of accesses play an important role, e. g., a blog post which the author does not desire to get viral.

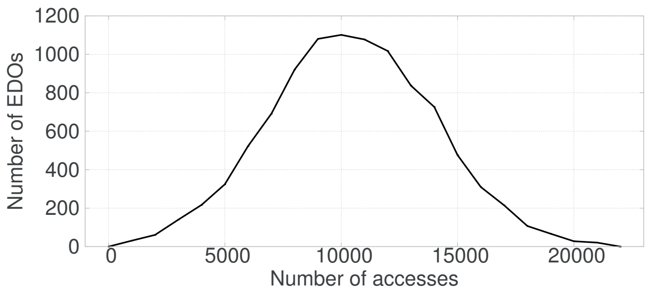


Figure 9: Standard deviation for the self destruction of EDOs.

The DNS infrastructure does not allow us to utilize or introduce a counter for the number of accesses to our publicly available EDOs. Therefore, we devised a probabilistic solution that could allow us to address this problem (Section 4.2.4). We next wanted to examine the accuracy of our approach. For this reason we created 10,000 EDOs and then we selected 10,000 to be a desired number of accesses. We then calculated the average number of accesses along with its standard deviation. Figure 9 depicts the number of accesses before an EDO is finally destroyed. More precisely, we found that each EDO was destroyed on average after 9928 accesses. The standard deviation value of our solution in terms of 3598 accesses (36% off from our assigned value) indicates that this approach can be selected as long as the number of accesses is acceptable as a rough estimate for the self-destruction of an object.

6.4 Manual Revocation Evaluation

Although our basic model is to revoke the access of publicly available data once the interest for it drops, there may exist cases where there is a pressing need for the immediate revocation of data. Although the concept behind of manual revocation is really simple, we nevertheless evaluated its effectiveness. To do so, we created 200 EDO documents and we divided them into two sets of 100 documents per set. In the first set we performed a manual revocation immediately after the creation of the encrypted documents while in the next set we performed the same action after they already had been accessed. This way we took into consideration (i) the case that an author of an EDO document wants to recall the access to the document right after it is been uploaded to a remote server, and (ii) the case that the document already have been accessed but the author does not desire any more publicity. As expected, in both cases we were able to completely destroy the decryption key in all 200 documents.

7. SECURITY ANALYSIS

We now analyze the presented scheme with respect to the security goal of retrospective privacy. Furthermore, additional attacks on the proposed system are discussed.

7.1 Retrospective Privacy

Retrospective privacy is fulfilled if an attacker is unable to access the contents of an EDO after its expiration. Under the assumption of a secure encryption system, this is possible in case the attacker manages to recover the decryption key of an object. Furthermore, we must take security issues of the DNS infrastructure into account.

7.1.1 Key recovery

Following the assumptions of Section 2.2, the attacker gains access to an encrypted EDO as well as the respective list of all destinations to where the key share was distributed after the object expired. Based on the DNS implementation, this results in a set of cache entries that previously represented the encryption key.

In an attack the expired and therefore still encrypted objects along with the list of key share domains are accessible to the adversary. At this point the information embedded in the acquired EDO can only be accessed if the attacker manages to recover the required amount of key bits from the list of queried domains for reconstructing the encryption key. The chance of successfully guessing the encryption key after expiration decreases over time.

Targeted guessing. Short after the expiration, a high number of cached 1-entries is still valid allowing the separation of 0- and 1-key bits based on an assumed threshold value: In the standard portrayal a threshold of $x = 2$ is optimal for the given error characteristics. Even though the correction capabilities for $e_{1\to0}$ errors can be increased by using larger portrayal lengths N, the x threshold must be at least 2 for providing the correction of $e_{0\to1}$ errors. An attacker therefore can assume $x = 2$. The number of valid 1 entries decreases over time, which increases the indistinguishability of key bits. Even though the threshold cannot be fulfilled correctly after some time, the existence of a 1-entry is more likely to be the remaining cache entry of a 1-bit of the key rather than a $e_{0\to1}$ flip error. Therefore targeted guessing may lead to success if a sufficient number of 1-caches still exist. Both, the possible assumption of the optimal x threshold as well as targeted guessing of entries highly increases the chance of an attacker to successfully recovering the encryption key. Nevertheless, the overall success highly depends on the amount of time that passed between the revocation of an object and the actual attack on the key information.

Time. Over time the number of remaining 1-entries in the cache decreases, which also reduces the probability for an attacker to successfully recover the encryption key from remaining entries. The period of time in which a successful attack is likely can be analyzed based on the frequency of common TTL values.

As soon as the expiration of one entry $TTL(c_{exp})$ leads to the destruction of the key bit, a set of remaining 1-entries $c_{rem} \in C_{rem}$ must have had the same or higher TTL values at the initialization of c_{exp}:

$$\exists c_{rem} \in C_{rem} : TTL(c_{rem}) \geq TTL(c_{exp}) \qquad (3)$$

That is, when c_{exp} was initially set and later triggered the destruction of the encryption key, all remaining entries that are also cached cannot have a TTL that is lower than that of c_{exp}. Due to this restriction two cases can be distinguished: In case (a) the majority of remaining entries has a TTL identical to c_{exp} and therefore expires at the same time. In this case the amount of information left for the attacker is insufficient for efficiently guessing the encryption key. In the second case (b) a majority of elements in C_{rem} has a TTL greater than the expired one. Based on the measurements on common TTL values, a lifetime of $86,400$ seconds (1 day) can be assumed for 58% of cache entries, the next greater TTL is $86,400 * 2$ and therefore provides another day to guess the remaining entries of the key. After that, the data is irrecoverably gone.

7.1.2 Security issues of the DNS infrastructure

Even though *Neuralyzer* utilizes the distributed and decentralized infrastructure provided by public DNS resolvers, several characteristics of this infrastructure can lead to security issues. Kührer et al. [20] found that in 2015 a significant amount (that is 20% of overall 6,753,748 resolvers in that specific measurement) of public resolvers run with BIND version 9.8.2, which can be manipulated through a remote code execution vulnerability. They also showed that a majority of public DNS resolvers are hosted by the top 25 networks wherein at least 20 offer telecommunication and broadband Internet services. Even though these results do not necessarily imply malicious behavior of service providers or the monitoring of traffic through a BIND vulnerability, the above can still lead to security issues for *Neuralyzer*. To overcome the threat of remote code execution, a fingerprinting of resolvers and the according software should be performed. When detecting a software version that can enable an attacker to perform monitoring of traffic on the public resolver, it should be excluded from the list of nodes that are considered in the portrayal of key bits.

In a scenario where the attacker is able to make use of the above issues, it must be assumed that the traffic of a fraction of resolvers of a portrayal can be monitored. This monitoring enables the attacker to save a history of cached/uncached entries during the lifetime of an object. In a retroactive attack this knowledge can help to reconstruct the encryption key from the remaining cache entries. As summarized in Section 7.1.1, even limited knowledge about the content of a portrayal can increase the probability of successfully guessing the related key bit dramatically. A simulation study on the amount of monitors required for learning about the encryption key shows that also smaller amounts of malicious resolvers are sufficient for reconstructing a significant portion of the key (see Appendix B for details). When extending the current filtering of resolvers to a selection of maximum distributed hosts of public resolvers, the threat of centrally controlled nodes can be reduced.

7.2 Further Attacks

Brute Force attack. A Brute Force attack on all possible DNS cache resolvers requires guessing of all potential destinations for bits of the key share. As summarized in EphPub [7] the number of possible entries exceeded 126 million in 2011 making a successful Brute Force key recovery unlikely for a DNS-based implementation. Moreover, the recovery of single key bits still requires the reconstruction of the full encryption key and a matching of this key to the explicit object it was applied to.

Sybil and infiltration attack. The Sybil attack, such as used by Zeng et al. [35] for showing vulnerabilities in the Vanish [14] system, is realized by controlling multiple virtual identities within the target infrastructure. Based on that the attacker is enabled to receive a large portion of the traffic occurring in a network allowing her to analyze the traversing packets. In context of a DNS-based implementation a Sybil cannot be realized through virtual nodes as there is no such thing as virtual identities for DNS resolvers. The required overhead for controlling the required amount of physical nodes in the DNS system makes a successful Sybil or infiltration attack therefore unlikely.

8. RELATED WORK

The automated deletion of data is not a new idea and researchers worked on this concept for quite a while. The first set of works involves trusted parties. Perlman proposed Ephemerizer, a trusted service that ensures timely expiration of emails [27, 28]. Nair et al. [25] extend the original idea of the Ephemerizer system through an identity-based cryptosystem. Systems such as X-Pire! [2] or the revocable backup system [6] also rely on a trusted key management server. Pöpper et al. [29] presented an approach utilizing porter devices for a secure storage of long term keys including explicit key deletion and forward-secret protocols, even under device compromise. Reimann et al. [30] proposed a revocation system that allows for long-term expiration dates of several months after publication. However, providing *flexible* expiration times of data using a *decentralized* infrastructure has not been proposed or investigated.

Vanish [14] was the first system not relying on a centralized, trusted system for the later revocation of data. Instead, it utilizes a decentralized architecture based on P2P distributed hash tables (DHT). Unfortunately, Vanish is susceptible to Sybil attacks [34] that can compromise the system by continuously crawling the DHT and saving each stored value before it expires. To overcome the vulnerabilities of Vanish, Zeng et al. [35] presented SafeVanish, which extends the length range of Vanish's key shares to substantially increase the attack cost, while it does also some improvement on the Shamir's Secret Sharing implemented in Vanish. In a survey about the vulnerabilities of self-destructing data systems, Geambasu et al. [13] implemented a framework for testing key-storage mechanisms based on different infrastructures and presented countermeasures to data-harvesting attacks. Casteluccia et al. [7] presented EphPub, a system that utilizes the DNS infrastructure and is capable of providing longer lifetimes for objects. Our major differences to EphPub is that we (i) designed a way for refreshing cache entries to prolong the lifetime of an object and (ii) applied access heuristics that manage the revocation of objects depending on the access behavior.

Another group of approaches is the cipher-text-policy attribute-based encryption (CP-ABE) schemes first introduced by Bethencourt [4]. With CP-ABE, access control policies can be enforced with attribute-based encryption where defined sets of credentials are required for encrypting sensitive data. These credentials are related to group or location attributes, e. g., specific enterprise departments, and included in a policy definition. Different to previous attribute-based encryption systems such as [11], attributes are used for a description of user credentials while access policies are defined by the encrypting party instead of storing this information in user keys. Hur et al. [17] presented a system providing the enforcement of access control policies even with revocation of attributes and thus access privileges. Balani and Ruj [3] took this concept to the cloud, outsourcing the decryption to a proxy server unable to retrieve information from the computations. Even though these approaches provide encryption-based handling of access policies they still cannot provide the dynamic revocation of files. That is the underlying crypto-infrastructure may be ported to a system such as the *Neuralyzer* while handling the key information must be performed in a decentralized manner.

9. CONCLUSION

In this paper, we proposed a novel approach for flexible revocation of online data. Other than recent work in this field, the central goal is to provide a metric-driven revocation mechanism that can be adapted to, e. g., the progression of user accesses over time instead of a predefined lifetime. To this end, the publicly accessible data is protected by encrypting it, encapsulating the encrypted content, and ensuring the destruction of the encryption key when the expiration time is reached. To assess our approach we created *Neuralyzer*, a proof-of-concept system based on the caching mechanism of public DNS resolvers that is able to refresh key information over time and expire its lifetime based on access novel heuristics. Results of a simulation study and experiments with a prototype implementation reveal that we can achieve flexible and reliable expiration times for the revocation of online data based on users' interest.

Acknowledgments

The research presented in this paper was supported by grant An-2014-0024 from MERCUR (Mercator Research Center Ruhr) and by the DFG Research Training Group GRK 1817/1.

10. REFERENCES

[1] O. Ayalon and E. Toch. Retrospective Privacy: Managing Longitudinal Privacy in Online Social Networks. In *Symposium on Usable Privacy and Security (SOUPS)*, 2013.

[2] J. Backes, M. Backes, M. Dürmuth, S. Gerling, and S. Lorenz. X-Pire!-A Digital Expiration Date for Images in Social Networks. *arXiv preprint arXiv:1112.2649*, 2011.

[3] N. Balani and S. Ruj. Temporal Access Control With User Revocation for Cloud Data. In *International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, 2014.

[4] J. Bethencourt, A. Sahai, and B. Waters. Ciphertext-Policy Attribute-Based Encryption. In *IEEE Symposium on Security and Privacy*, 2007.

[5] M. Bishop, E. R. Butler, K. Butler, C. Gates, and S. Greenspan. Forgive and Forget: Return to Obscurity. In *New Security Paradigms Workshop*, 2013.

[6] D. Boneh and R. Lipton. A Revocable Backup System. In *USENIX Security Symposium*, 1996.

[7] C. Castelluccia, E. De Cristofaro, A. Francillon, and M.-A. Kaafar. EphPub: Toward Robust Ephemeral Publishing. In *IEEE International Conference on Network Protocols (ICNP)*, 2011.

[8] C. Conley. The Right to Delete. In *AAAI Spring Symposium: Intelligent Information Privacy Management*, 2010.

[9] J. Daemen and V. Rijmen. *The Design of Rijndael: AES – the Advanced Encryption Standard*. Springer Science & Business Media, 2002.

[10] E. De Cristofaro, C. Soriente, G. Tsudik, and A. Williams. Hummingbird: Privacy at the Time of Twitter. In *IEEE Symposium on Security and Privacy*, 2012.

[11] T. Eissa and G.-H. Cho. A Fine Grained Access Control and Flexible Revocation Scheme for Data Security on Public Cloud Storage Services. In *International Conference on Cloud Computing Technologies, Applications and Management (ICCCTAM)*, 2012.

[12] European Commission. Factsheet on the "Right to Be Forgotten" Ruling, C-131/12. http://ec.europa.eu/justice/data-protection/files/factsheets/factsheet_data_protection_en.pdf, 2014.

[13] R. Geambasu, T. Kohno, A. Krishnamurthy, A. Levy, H. Levy, P. Gardner, and V. Moscaritolo. New Directions for Self-Destructing Data Systems. Technical report, University of Washington, 2011.

[14] R. Geambasu, T. Kohno, A. A. Levy, and H. M. Levy. Vanish: Increasing Data Privacy with Self-Destructing Data. In *USENIX Security Symposium*, 2009.

[15] R. Gross and A. Acquisti. Information Revelation and Privacy in Online Social Networks. In *ACM Workshop on Privacy in the Electronic Society (WPES)*, 2005.

[16] Huffington Post. Experts Say Facebook Leak of 6 Million Users' Data Might Be Bigger Than We Thought. http://www.huffingtonpost.com/2013/06/27/facebook-leak-data_n_3510100.html, Jun 2013.

[17] J. Hur and D. K. Noh. Attribute-Based Access Control With Efficient Revocation in Data Outsourcing Systems. *IEEE Transactions on Parallel and Distributed Systems*, 22(7):1214–1221, 2011.

[18] Internet Live Stats. Total Mumber of Websites. http://www.internetlivestats.com/total-number-of-websites/, Aug 2015.

[19] B. Krebs. Online Cheating Site AshleyMadison Hacked. http://krebsonsecurity.com/2015/07/online-cheating-site-ashleymadison-hacked/, Jul 2015.

[20] M. Kührer, T. Hupperich, J. Bushart, C. Rossow, and T. Holz. Going Wild: Large-Scale Classification of Public DNS Resolvers. In *ACM SIGCOMM Internet Measurement Conference (IMC)*, 2015.

[21] M. Madejski, M. L. Johnson, and S. M. Bellovin. The Failure of Online Social Network Privacy Settings. Technical report, Columbia University, 2011.

[22] C. D. Marsan. 15 Worst Internet Privacy Scandals of All Time. http://www.networkworld.com/article/2185187/security/15-worst-internet-privacy-scandals-of-all-time.html, Jan 2012.

[23] Mashable. 98,000 Hacked Snapchat Photos and Videos Posted Online. http://mashable.com/2014/10/13/the-snappening-photos-videos-posted, Oct 2014.

[24] P. V. Mockapetris. RFC 883, Domain Names – Implementation and Specification. 1983.

[25] S. K. Nair, M. T. Dashti, B. Crispo, and A. S. Tanenbaum. A Hybrid PKI-IBC Based Ephemerizer System. In *New Approaches for Security, Privacy and Trust in Complex Environments*, 2007.

[26] P. Papadopoulos, A. Papadogiannakis, M. Polychronakis, A. Zarras, T. Holz, and E. P. Markatos. K-Subscription: Privacy-Preserving Microblogging Browsing Through Obfuscation. In *Annual Computer Security Applications Conference (ACSAC)*, 2013.

[27] R. Perlman. File System Design With Assured Delete. In *IEEE International Security in Storage Workshop (SISW)*, 2005.

[28] R. Perlman. The Ephemerizer: Making Data Disappear. *Journal of Information System Security (JISSec)*, 1:51–68, 2005.

[29] C. Pöpper, D. Basin, S. Čapkun, and C. Cremers. Keeping Data Secret Under Full Compromise Using Porter Devices. In *Annual Computer Security Applications Conference (ACSAC)*, 2010.

[30] S. Reimann and M. Dürmuth. Timed Revocation of User Data: Long Expiration Times From Existing Infrastructure. In *ACM Workshop on Privacy in the Electronic Society (WPES)*, 2012.

[31] D. Rosenblum. What Anyone Can Know: The Privacy Risks of Social Networking Sites. *IEEE Security & Privacy*, (3):40–49, 2007.

[32] The Register. iCloud Fiasco: 100 Famous Women Exposed Nude Online. http://www.theregister.co.uk/2014/08/31/jlaw_upton_caught_in_celeb_nude_pics_hack, Aug 2014.

[33] Wisemetrics. Your Tweet Half-Life Is 1 Billion Times Shorter Than Carbon 14's. http://blog.wisemetrics.com/tweet-isbillion-time-shorter-than-carbon14/, Mar 2014.

[34] S. Wolchok, O. S. Hofmann, N. Heninger, E. W. Felten, J. A. Halderman, C. J. Rossbach, B. Waters, and E. Witchel. Defeating Vanish With Low-Cost Sybil Attacks Against Large DHTs. In *ISOC Network and Distributed System Security Symposium (NDSS)*, 2010.

[35] L. Zeng, Z. Shi, S. Xu, and D. Feng. SafeVanish: An Improved Data Self-Destruction for Protecting Data Privacy. In *International Conference on Cloud Computing Technology and Science (CloudCom)*, 2010.

APPENDIX

A. COMPARISON OF CORRECTION CAPABILITIES

In the following our portrayal **(P)** is compared to the Reed-Solomon **(RS)** and Golay **(G)** error correcting code with respect to each scheme's individual overhead. To do so, a fixed number of errors $T = 1000$ and key length $z = 128$ are defined that should be corrected by a scheme with minimum possible overhead.[2] For a binary input of length z the above schemes can correct the following number of errors in general:

$$\textbf{RS} \begin{cases} \frac{n-k}{2} \cdot \lceil \frac{z}{k \cdot m} \rceil & : n-k \text{ even} \\ \frac{n-k-1}{2} \cdot \lceil \frac{z}{k \cdot m} \rceil & : n-k \text{ odd} \end{cases}, \tag{4}$$

where $n = 2^m - 1$ with $7 \leq n \leq 2^{16} - 1$ is the codeword length, $3 \leq m \leq 16$ is the input word length, and $k < n$ is the number of words to be encoded in one code word.

$$\mathbf{P} : z(N - x - 1), \tag{5}$$

where N is the length of a portrayal, and x is the correction threshold. To be robust against flipping of bits in both directions, the total correction capability for $e_{0\to1}$ must be at least 1. This requirement is fulfilled for $x \geq 1$ and $N \geq 3$.

The parameters for **G** are fixed and allow up to 3 corrections for the extended binary $G[24, 12, 8]$.

A **RS** code is optimal only for non-binary input and can be adapted to encoding the binary encryption key by setting $m = 8$ for representing 1 byte per word. Under this assumption the overhead and correction capability of **RS** (assumed $n - k$ even) is as follows for $T = 1000$:

$$\begin{aligned} T &\leq \frac{n-k}{2} \cdot \lceil \frac{z}{k \cdot m} \rceil & (6) \\ \Rightarrow 1000 &\leq \frac{255-k}{2} \cdot \lceil \frac{128}{k \cdot 8} \rceil | \cdot 2 & (7) \\ \Leftrightarrow 2000 &\leq 255 - k \cdot \lceil \frac{128}{k \cdot 8} \rceil | k = 1 & (8) \\ \Rightarrow 2000 &\leq 2032 & (9) \end{aligned}$$

To correct at least $T = 1000$ errors, **RS** must be applied with $m = 8$, $n = 255$, $k = 1$ leading to a total overhead of factor 16.

For any T and an input length $z = 128$ the overhead of **RS** is as follows:

$$2T \leq (n^m - 1 - k) \cdot \lceil \frac{128}{m \cdot k} \rceil \tag{10}$$

[2]In context of a scheme's correction capabilities the overhead defines the amount of additional bits required for a code that is capable of correcting the fixed error rate.

For a minimum threshold of $x = 2$, **P** provides the following overhead for $T = 1000$:

$$\begin{aligned} T &\leq z(N - x - 1) & (11) \\ \Rightarrow 1000 &\leq 128(N - 2 - 1) & (12) \\ \Leftrightarrow 1000 &\leq 128N | N = 8 & (13) \\ \Rightarrow 1000 &\leq 1024 & (14) \end{aligned}$$

This leads to a parameter set of $N > 8, x = 2$ and a total overhead of factor 9 for the portrayal scheme. The correction capability for any error rate T for an input of length $z = 128$ is as follows:

$$\frac{T}{128} \leq N \tag{15}$$

For providing the same minimum correction rate with **G**, the input length must be extended: to correct at least $T = 1000$ errors, an input length of 334 is required leading to a total overhead of factor 63.

B. CONTROL OF M PUBLIC RESOLVERS

Under the assumption of an attacker who is able to utilize either a remote code execution vulnerability in the software of a resolver or a malicious service provider hosting a fraction of resolver, a relative amount of m entries, that is M entries in total, in the key representation is known by the attacker. To analyze the effects of such an attack on the security of *Neuralyzer*, two cases must be distinguished (for the following we assume a maximum number of resolvers used in the portrayal: each domain in the key portrayal is organized by exactly 1 resolver):

In the best case (bc) scenario (from a user's perspective) the number of different key bits affected by the attack is minimal. That is for the portrayal C_{IxN} the number of rows I is minimal while the number of columns N is maximal. For a key length I and portrayal size N the probability of being attacked in a best case distribution of M controlled domains is as follows:

$$Pr(bc) = \binom{I \cdot N}{M} \tag{16}$$

Opposed to the best case scenario the attacker has the maximum possible knowledge about the encryption key in case the controlled resolvers are distributed over a maximum number of rows I in the portrayal. We simulated this attack for different fractions m of controlled resolvers and varying key lengths I and portrayal sizes N. Results show that even for small m the attacker can gather a significant amount of information about the encryption key. For $m = 0.05$ an attacker can control from 14% (at $I = 128$, $N = 3$) up to 40% (at $I = 128$, $N = 10$) of rows in the key portrayal.