# Forgetting with Puzzles: Using Cryptographic Puzzles to support Digital Forgetting

Ghous Amjad
Brown University & NYUAD
Providence, Rhode Island, USA
ghous_amjad@brown.edu

Muhammad Shujaat Mirza
NYUAD
Abu Dhabi, UAE
muhammad.mirza@nyu.edu

Christina Pöpper
NYUAD
Abu Dhabi, UAE
christina.poepper@nyu.edu

## ABSTRACT

Digital forgetting deals with the unavailability of content uploaded to web and storage servers after the data has served its purpose. The content on the servers can be deleted manually, but this does not prevent data archival and access at different storage locations. This is problematic since then the data may be accessed for unintended or even malicious purposes long after the owners have decided to abandon the public availability of their data. Approaches which assign a lifetime value to data or use heuristics like interest in data to make it inaccessible after some time have been proposed, but digital forgetting is still in its infancy and there are a number of open problems with the proposed approaches.

In this paper, we outline a general use case of cryptographic puzzles in the context of digital forgetting which—to the best of our knowledge—has not been proposed or explored before. One problem with recent proposals for digital forgetting is that attackers could collect or even delete anyone's public data during their lifetime. In our approach, we deal with these problems by making it hard for the attacker to delete large quantities of data while making sure that the proposed solutions will not adversely deteriorate user experience in a disturbing manner. As a proof-of-concept, we propose a system with cryptographic (time-lock) puzzles that deals with malicious users while ensuring the permanent deletion of data when interest in it dies down. We have implemented a prototype and evaluate it thoroughly with promising results.

## CCS CONCEPTS

• **Security and privacy → Privacy-preserving protocols**; **Privacy protections**;

## KEYWORDS

Digital Forgetting; Cryptographic Puzzles; Time-lock Puzzles

**ACM Reference Format:**
Ghous Amjad, Muhammad Shujaat Mirza, and Christina Pöpper. 2018. Forgetting with Puzzles: Using Cryptographic Puzzles to support Digital Forgetting. In *CODASPY '18: Eighth ACM Conference on Data and Application Security and Privacy, March 19–21, 2018, Tempe, AZ, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3176258.3176327


*CODASPY '18, March 19–21, 2018, Tempe, AZ, USA*

ACM ISBN 978-1-4503-5632-9/18/03…$15.00
https://doi.org/10.1145/3176258.3176327

## 1 INTRODUCTION

Users share tremendous amounts of data by uploading content on social media: in 2017, users have been reported to have shared 46,000+ photos on Instagram and watched 4,000,000 videos on Youtube, they have published 75,000 pieces of content on Tumblr and have sent roughly 456,000 tweets on Twitter—all *per minute* [4]. Typically, the content uploaded to social networks and sharing platforms, such as photos, videos, and messages, has relevance only for a certain amount of time and loses importance when time passes. The data, though, often remains available, potentially open to misuse by third parties out of curiosity or with malicious intent.

Content owners and persons related to the content may not want the data to remain publicly accessible after a certain amount of time has passed since the availability of past data might be damaging for them in present or in the future. The Right to be Forgotten [15] and ephemerality [21] are concepts that are gaining momentum in digital contexts with the passage of time. However, when something is deleted from the Internet there is no guarantee that the content is indeed gone for good. The data might have been shared, duplicated, or archived elsewhere online since the time of its creation and users generally do not have control over this process.

To deal with these issues, technical proposals with the purpose of *Digital Forgetting* have been made, including Vanish [7], EphPub [3], and Neuralyzer [28]. They try to achieve their goals by uploading the content in encrypted form along with information on where and how to extract the decryption key during the data lifetime. Since trust in a central third party is not a generally applicable assumption, these approaches store the decryption key on a public distributed infrastructure, such as in distributed hash tables [7], on domain name system servers [3, 28], or using public websites [19]. The main property that is desired by such an infrastructure is that it 'forgets' the key with time and that content owners are also able to manually destroy the key when needed. The system and attacker model that these approaches deal with is one where the attacker becomes interested in the data after its expiration. Due to the deletion of the key by then, the attacker will be unable to access the unencrypted content.

In this paper, we build on these approaches and focus on impeding and preventing an attacker from interfering with the data before its expiration time. This is challenging because it concerns a time when data is meant to be publicly available. We concentrate on the part where an attacker may want to proactively collect large amounts of content or remove it by interfering with the key or deleting it on the public infrastructure. Knowing that an attacker

can pretend to be a normal user that should be granted access to public data during its lifetime, our goal is to make it as difficult and costly for an attacker to interfere or access data *on a large scale* before the expiration time. The solution should affect normal users as little as possible and still be effective against a strong adversary with strong computing and storage power.

With these goals in mind, we introduce the idea of applying *cryptographic puzzles* to support distributed schemes for digital forgetting by securing them against attacks that aim to destroy or collect data on a large scale. Our basic idea is that everybody accessing the data, including future adversaries, needs to provide a proof of work before being able to access the content of an encrypted object. We achieve this without a central server for work verification but instead only use the distributed infrastructure and *implicit verification with time-lock puzzles* [20], given at the moment when a participant is able to decrypt the data. This ensures that if an attacker has access to the underlying data, he or she indeed has done some work. Time-lock puzzles also have the property of being sequential in nature, which means using more machines will not reduce the time to their solution. We show that it is hard for an attacker to successfully delete the decryption key and collect data on a large scale when addressing the system as a whole, in particular a large percentage of all objects protected by a scheme for digital forgetting.

As proof of concept, we present a system that incorporates puzzles with the recent Neuralyzer [28] and show that introducing cryptographic puzzles substantially reduces the impact of an attacker that attempts to delete or collect a lot of data. In more details, we design and evaluate a scheme in which a normal user gets to solve one easier, faster-to-solve puzzle, which takes little time. An attacker, however, will have to go through the process of solving two puzzles, the second one being more time consuming, before data can be deleted (we can have more than two puzzles of increasing difficulty as long as this does not increase setup time significantly). Our evaluation results demonstrate that our proposal can be realized and the extra time needed for creating the puzzles is minimal. The extra overhead in terms of file sizes is small and it is independent of the data type and size. The time to break the puzzle to gain data access varies smoothly with the selected difficulty of the puzzle.

In short, our main contributions are as follows:

- We revise and extend previous adversarial models that have been proposed for digital forgetting to incorporate attacks during the data lifetime.
- We propose the use of cryptographic puzzles for digital forgetting and demonstrate how they can be applied in a distributed manner without trust in one central verification party.
- We design a scheme that enables access to public data at very low cost for regular users but make it very costly (time-consuming) for an attacker to save or delete decryption keys on a large scale.
- We develop a prototype implementation based on Neuralyzer [28]. Our results show that the overhead in terms of the time needed for data object creation and the increase in file size are minimal. The time to access data for a normal user can flexibly be adjusted with the difficulty of the puzzle.

## 2 MODELS AND RESEARCH GOALS

A number of proposals have been made to assign a pre-determined lifetime value to published data [3, 7, 19] or to use heuristics to make the data inaccessible [28]; heuristics may, e. g., be based on the interest in data as determined by the number of recent access operations. In the following, we introduce the considered system and attacker models, describe our research goals, and give background information on one specific system we will base our evaluations on.

### 2.1 System Model

The general procedure of all schemes above is as follows:

(1) The owner of the data content creates an encrypted data object and assigns a (preliminary or fixed) expiration time $t_e^0$ for the object. We denote the time at which the encrypted data content is published or uploaded for public access as $t_c$.
(2) During the lifetime of the data object, i. e., at all times $t$ with $t_c \leq t < t_e^x$, anyone can access the object and can successfully decrypt it to access the data. $x$ here represents the total amount of lifetime which may have been extended from the originally assigned lifetime of the object: $t_e^x \leftarrow t_e^0 + x$. If no lifetime extensions were made then $x = 0$ and $t_e^x = t_e^0$.
(3) After $t_e^x$, no one should be able to access the data object other than the people who already know the decryption key.

We consider systems with a distributed, dynamic, and publicly accessible infrastructure where data—in particular decryption keys—can be temporarily stored. This does not need to be an infrastructure with the primary purpose of data storage (like cloud servers) but rather comprises a secondary storage system where data is stored in ephemeral storage (such as cache entries); examples include entries in distributed hash tables (DHTs) [7], website encodings [19], and domain name server (DNS) caches [3, 28] (Sec. 2.4 & App. A.1 provide details). We do not alter the infrastructure itself, but make use of states or storage space to upload and retrieve keys during their lifetimes. The dynamics of the infrastructure lead to modifications (churn) and data loss that result in automatic deletion of the stored data (keys) after some time. The solutions we consider place no trust in a third-party server.

The encrypted data objects from a digital forgetting scheme comprise the encrypted content as well as the information required to successfully retrieve and/or build the required decryption key. The successful retrieval of the key is only possible if the object has not yet expired and is still accessible. Once keys are uploaded to the ephemeral storage, they typically cannot be modified directly (at least for DHTs and DNS cache entries).

### 2.2 Attacker Model

In our threat model we consider attackers whose target is not a single user but rather system-wide attacks that attempt to collect or interfere with data on a large scale for many users. In a sense these attackers are similar to perpetrators in a network-wide denial-of-service attack as they want to cause damage to millions of users

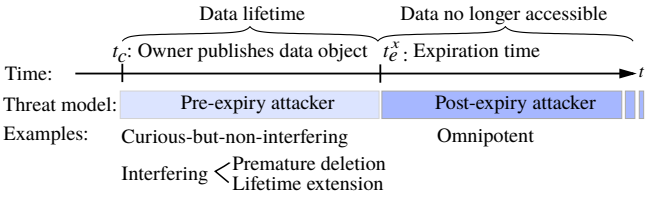


**Figure 1: Considered attacker model: Before the data expiration, we consider curious-but-non-interfering attackers and interfering attackers. After the expiration time, we consider omnipotent attackers that may also attempt to attack targeted users/data objects.**

indiscriminately. That means, our threat model does not focus on targeted attacks on specific users before the expiration time, but includes attackers that want to collect or delete lots of data during the data lifetime; a specific user and a data object may turn into a focused attack target only after expiration of said object.

This threat model is novel in the sense that, in previous schemes, a user could be turned into an attack target by an adversary only *after* the content of interest had expired as it was assumed that the attacker becomes interested in a specific data object after its expiry. Our new attacker may have the following malicious intents (see Fig. 1):

(1) The *curious-but-non-interfering* attacker may want to proactively collect a large amount of data indiscriminately once available in order to extract data of interest and use it for malicious purposes after its expiration. Such an attacker is looking to take a snapshot of all the publicly available data in some fixed time period and can keep repeating this regularly, e. g., daily, weekly, or monthly. Taking such snapshots is feasible for a strong adversary where its success and extent depend only on the size of the available communication bandwidth and size of the storage space. This attacker essentially defeats the essence of digital forgetting.
(2) The *interfering* attacker may want to make (large amounts of) data inaccessible during the data lifetime by launching an attack that exploits the weakness of the proposed schemes, deleting decryption keys during the lifetimes and thus making data objects inaccessible. A second instance of an interfering attacker would keep data alive by extending their lifetimes.

Curious-but-non-interfering attackers are threats to all proposals for digital forgetting [3, 7, 19], while interfering attackers are an issue mainly to schemes that allow flexible lifetime extensions (e. g., based on user interest [28]).

## 2.3 Goals and Challenges

Although we may not be able to prevent attacks entirely due to the intended public and unrestricted availability of the data during its lifetime, our goal is to make sure that these attacks and actions become costly enough to deter attackers. This evokes a need for making an attacker (and common users) do work before they are able to save or delete a data object. The work should be of such a nature that over time, once an attacker has gone through a certain number of objects, it should not take less time than before to access or delete the next object.

A few additional thinkgs need to be kept in mind: Having many machines should not make the process of getting to one object faster; if that happens any proposed scheme would have very little impact for resource-rich attackers. Another important thing to consider is that we should have strong control over how much work a normal user on a normal machine (or mobile device) will need to do so that they do not face difficulty in obtaining the requested data.

With respect to our system model, we can thus formalize our goals as follows:

**Functional Goal:** Between data object creation time $t_c$ and the expiration time $t_e^x$, anyone who has the (encrypted) data object can access its content, including curious-but-non-interfering attackers and interfering attackers.

**Security Goal 1:** An attacker who becomes interested in a specific data object (for whatever reason) after $t_e^x$ (after its expiration), should not be able to access it.
**Security Goal 2:** A curious-but-non-interfering attacker should find itself in a situation where it is infeasible to take a snapshot of a meaningful percentage of publicly accessible content (data objects between $t_c$ and $t_e^x$) in a fixed time period, e. g., a day.
**Security Goal 3:** An interfering attacker should find itself in a situation where a deletion attack over a meaningful percentage of accessible data is infeasible.

The first security goal concerns only specific data objects whereas the latter two security goals are concerned with large sets of publicly available data objects. Our **notion of security** is as follows: Suppose an attacker could take a snapshot or delete tens of thousands of data objects (some percentage of the total available public content) in a fixed time period with one (powerful) machine. If we can increase the number of machines required to pull off such an attack to a number which increases the costs for an attacker in terms of power or rent for the machines by a huge percentage and thereby reduces the amount of data objects captured in a snapshot or deleted in a fixed time period, we say a large-scale attack has now become infeasible for the attacker.

## 2.4 Background on Neuralyzer Scheme

As our proposed system is designed to support the existing distributed schemes for digital forgetting, this section serves as an overview of one such scheme: Neuralyzer [28]. We will be referring to this scheme later in the paper to evaluate the effectiveness of incorporating cryptographic puzzles against attacks that aim to destroy or collect data on a large scale.

In Neuralyzer [28], similar to EphPub [3], data is encrypted and stored along with a list of domains on DNS server (resolver) caches that encode the decryption key. This object is called Ephemeral Data Object (EDO). To encode the key in a list of resolver-domain pairs, the following steps are taken:

(1) Per key bit, a domain and a DNS resolver are chosen at random. The domain is selected from a list of domains that are used infrequently (i. e., not part of the Alexa list). To add redundancy and enable key lifetime extensions, multiple resolver-domain pairs are selected per key bit.

(2) It is checked whether that domain is in the server's cache by sending the server a *non-recursive* request.
(3) If it is uncached, the resolver-domain pair is selected to be used. If the pair is meant to represent a key bit of value 0 then it is kept the way it is, otherwise (for key bits 1) a *recursive* DNS request is sent to the resolver so that the domain is stored in the server's cache.

This allows to encode the key bits to be temporarily stored in distributed DNS resolver cache entries and the key bits to be extracted as long as they are present in the cache. The decryption process is as follows:

(1) For each key bit, several resolver-domain pairs are stored. Each resolver is sent a non-recursive request about the respective domain. If more than a threshold of domains are present in the caches of their respective servers, the key bit is interpreted as 1, otherwise as 0.
(2) Now for each key bit representing 1, one of its resolver-domain pairs is selected and a recursive request is sent if the TTLs (time-to-lives) of all these pairs fulfill a certain criteria. This is done in order to extend the lifetime of the EDO. Hence, the key only gets deleted either by manual revocation or when the interest in its underlying data dies down. The lifetime extension is called 'refreshing' the EDO.

Note that manual revocation is done in a similar fashion as lifetime extension. For each key bit 0, its resolver-domain pairs are selected and recursive requests are sent.

# 3 USE OF CRYPTOGRAPH. PUZZLES

Cryptographic puzzles are a well-known defense strategy against distributed denial-of-service (DDoS) attacks [10, 27]. In these attacks many machines send simultaneous requests to a particular server, which overwhelms the server and makes the service unavailable to its intended users. Cryptographic puzzles are used to prevent such attacks; they can be time-based [20], memory-based [5], or bandwidth-based [24]. When an attacker sends a request, the server replies with a question or puzzle (see Fig. 2a). To answer that question, an attacker will have to perform considerable computations or use substantial memory. The server then verifies the answer and serves the data. Now for a normal user the amount of time spent on solving the puzzle is negligible but an attacker would need to solve thousands of puzzles to make the DDoS attack successful which will cost him a lot of time and power.

We argue that a defense against the collection or deletion of large amounts of data in digital forgetting can be similar to defending against a network-based denial-of-service attack. In our case, the goal is to make it difficult for attackers to delete each data object by making them solve puzzles before they get access to the objects.

As a difference, we are forced to use *self-decrypting puzzles* in the sense that once the attacker has the data object, there will be no communication with the data uploader or any third party server for the purposes of verifying the solution of the puzzle (we stress that we are addressing data that is supposed to be publicly available at some point in time—so we cannot make use of classical access-control mechanisms to restrict the access to the data).

Our puzzles and data are contained inside a data object and the data should only be accessible after the puzzle is solved correctly. Proposals involving cryptographic puzzles and proof of work typically involve three entities: the system that creates the puzzle, the one which solves it, and then the one that verifies it. Typically the verifier is a third party or the creator of the puzzle itself. In our setting, we can neither resort to a third party nor add more communication to the system (see Figure 2b).

## 3.1 The Time-Lock Puzzle

Different cryptographic puzzles are commonly used. They are, e. g., based on the factorization problem (factorizing the product of two large primes) or on finding a string such that the hash of the string begins with a certain number of zeros [1] (as used in the mining algorithm of current digital currencies). Puzzles like hashcash can be solved faster if the solvers have many machines at their disposal.

We base our proposal on the time-lock puzzle proposed by Rivest et al. [20]. Its security depends on the hardness of the factorization problem. The main idea behind the puzzle is making sure that an attacker must spend a certain amount of time on the puzzle before getting the data. The major strength of this puzzle is the fact that it is intrinsically sequential in nature and having more computers will not help in getting to the solution faster. The time-lock puzzle is based on repeated squarings which are not parallelizable. We describe it first for encrypting a general message $m$ and then demonstrate its applicability to digital forgetting.

*3.1.1 Creating the Puzzle.* We want to encrypt a message $M$ such that its decryption takes at least a period of $T$ seconds. Choose two large random primes $p$ and $q$. Let $n$ be their product and $\Phi(n) = (p-1)(q-1)$. Let $S$ be the squarings modulo $n$ per second a computer can perform. Call $t$ the product of $T$ and $S$. Generate a random key $K$ which should be big enough that brute-force searching for it is infeasible. Encrypt the message $M$ with $K$ to produce the ciphertext $C_m$. Pick a number $\alpha \bmod n$ and encrypt the key $K$ as $C_k = K + \alpha^{2^t} \bmod (n)$. To do this efficiently, first compute $e = 2^t \mod (\Phi(n))$ and then $b = \alpha^e \mod (n)$. Forget everything and only keep $n$, $\alpha$, $t$, $C_m$, and $C_k$.

*3.1.2 Solving the Puzzle.* As searching for the key $K$ is infeasible, the fastest approach is to compute $b \mod (n)$. Knowing $\Phi(n)$ will make computing $e$ efficient and hence $b$ will be computed easily too, but to get $\Phi(n)$ from $n$ is as hard as factoring $n$, which is far less efficient than repeated squaring. So the fastest way of computing $b$ is to start with $\alpha$ and square the result $t$ times sequentially. To control the difficulty of the puzzle we can toggle $t$; the bigger $t$, the more time it will take to get to the solution.

## 3.2 Puzzles for Digital Forgetting

In digital forgetting, we deal with content available online to which we add the property of becoming forgotten. Online data needs to be accessed quickly by a normal user. When we add puzzles, it means that more time will be needed to access the object as a user will spend some time on solving the puzzle. So we need puzzles which can be solved in an acceptable time.

Our major insight is that we can add more than one puzzle to the same data object in such a way that a malicious user will need to solve all of the puzzles to be able to launch deletion attacks while a normal user only accessing the data needs to solve only as little

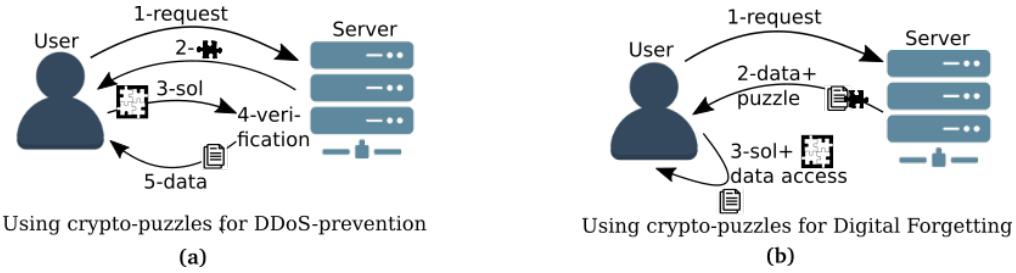


**Figure 2: Use of Crypto Puzzles. (a) The parties and communication involved in DDoS prevention based on cryptographic puzzles: 1) The user requests data, 2) the server responds with a cryptographic puzzle, 3) the user solves it and returns the solution, 4) the server verifies the solution and 5)—given successful verification—transmits the requested data. (b) The setup and communication involved in our use of cryptographic puzzles for digital forgetting: 1) The user requests the data, 2) obtains a data object that contains the data protected by a cryptographic puzzle, and 3) can access the data after having successfully solved the puzzle.**

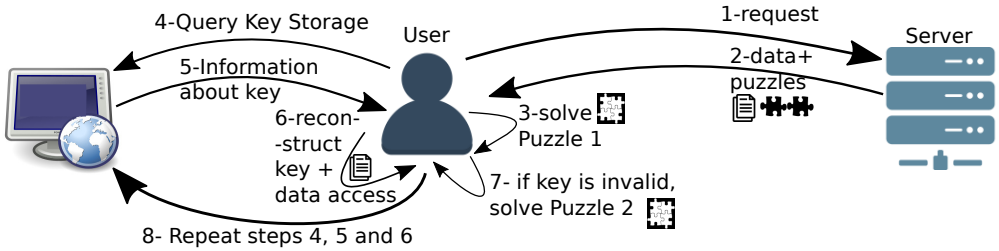


**Figure 3: Our basic design: 1) The user requests data, 2) the server responds with a data object containing two timelock puzzles and encrypted data content, 3) the user solves the first puzzle and gets access to the decryption key retrieval information, 4) the user uses this information to retrieve the key by querying the distributed key storage, 5) the user gets a response from the key storage, 6) the user constructs the key and decrypts the data content. 7) If the reconstruction was not successful (e. g., due to an interfering attacker), the user solves the second puzzle, and 8) the user repeats steps 4 to 6 using the new key retrieval information.**

as possible, thus introducing a form of asymmetry between regular users and active attackers. Since the puzzles can be of increasing difficulty, a framework of puzzles is being put to use; it will be explained in detail in the next section.

For a realization of our proposal we need to select parameters for the time-lock puzzle (for concrete instantiations in our implementation, see Section 5). Typically we consider a puzzle easy if can be solved by a standard PC or laptop within a few seconds. We should thus set $t$ to a value that allows one puzzle to be so solved within this timeframe (depending on the assumed number of squarings per second that can be calculated). We give typical values when reporting our evaluation and results.

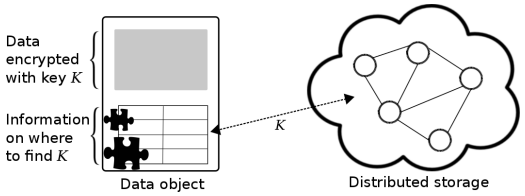


**Figure 4: Idea: The public data object contains the relevant data in encrypted form. The encryption key $K$ is stored in a dynamic distributed storage that automatically deletes the key after a while. The data object also contains information on where to find $K$ during its lifetime; we assume it is stored in a table with (<key bits>, <address>)-tuples. Our proposal protects the table by cryptographic puzzles.**

## 4 SYSTEM DESIGN

Our proposed system design is depicted and explained in Figure 3. Before we present details of our proposal, we shortly review the general concept of schemes proposed for digital forgetting with and without the capability to update and reset data lifetimes.

### 4.1 Preliminaries

We consider a public data object in which data is encrypted with an encryption key $K$. This data object additionally contains information on where to find the key during the data lifetime, see Figure 4. Without loss of generality, we assume that the encryption key $K$ is split into one or more key parts (key shares or key bits) $K_i$ that

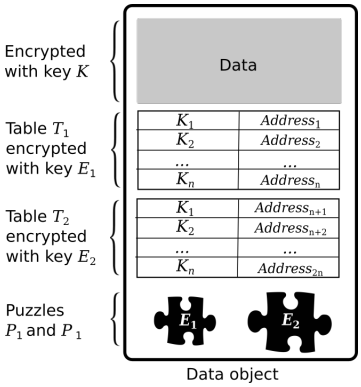


**Figure 5: Structure of a digital-forgetting data object, consisting of the data itself and two tables for key-lookup, each encrypted with a key protected by a puzzle of increasing difficulty.**

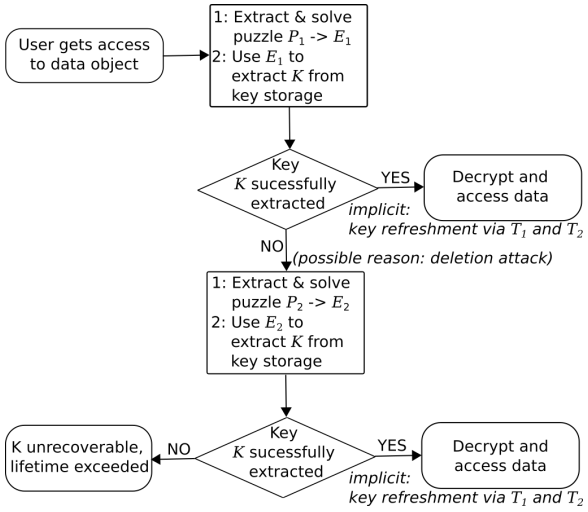


**Figure 6: Process and steps in using our puzzle-based structure.**

are stored by the data publisher/creator on a dynamic distributed storage that automatically and gradually deletes the $K_i$ after a while. We assume that the information on where to find the key bits is stored in a table $\mathcal{T}$ with (<key bits>, <address>)-tuples. The key is generally recoverable as long as the key bits are available and extractable from the infrastructure. The scheme may provide support for lifetime extension, e. g., based on user interest in the data: as long as there is sufficient interest, the key will remain available in the infrastructure. These lifetime extensions can be realized in a manner that is fully transparent to the users, but they may also enable an attacker to interfere with the intended lifetimes in a malicious manner.

## 4.2 Our Approach

Our scheme adds cryptographic time-lock puzzles to protect the access information table $\mathcal{T}$ in the data object, thus adding a layer of difficulty for the key extraction process. The idea is to encrypt $\mathcal{T}$ using a symmetric key $E$, protect $E$ by a time-lock puzzle, and make the resulting puzzle part of the data object. To access the information on where to find the key bits $K_i$ will require the user (and any potential attacker) to solve a puzzle first, thus extracting $E$, which will then reveal the table $\mathcal{T}$ with the storage locations of $K_i$. The puzzle itself does not have an expiration time, but the table that can be accessed with the extracted key $E$ will only contain valuable information during the lifetime of the data object.

More precisely and in order to make the premature deletion of data more difficult than data access during the data lifetime, our scheme consists of the use of *two* tables $\mathcal{T}_1$ and $\mathcal{T}_2$, each protected by a time-lock puzzle of different difficulty level, see Figure 5. The tables $\mathcal{T}_i$, $i \in \{1, 2\}$, contained in each data object have the following properties:

(1) $\mathcal{T}_1$ and $\mathcal{T}_2$ provide access information to separate storage locations of the key $K$. No matter which table (and puzzle) a receiver uses to access the decryption key $K$, (s)he will be able to get the data before its expiry date.

(2) $\mathcal{T}_1$ is protected by an easier puzzle $\mathcal{P}_1$, $\mathcal{T}_2$ by a harder puzzle $\mathcal{P}_2$ that takes more time to solve. The first puzzle $\mathcal{P}_1$ provides a hurdle against large-scale data collection attacks. The second puzzle $\mathcal{P}_2$ provides protection against data deletion attacks in which an attacker tries to remove/invalidate key bits during the data lifetime.

(3) If the key bits $K_i$ accessible by the information in table $\mathcal{T}_1$ get inaccessible as a result from a deletion attack, the data can still be accessed by recovering the key $K$ from table $\mathcal{T}_2$. If the key bits $K_i$ accessible by the information in table $\mathcal{T}_1$ get inaccessible as a result of other natural reasons near the object's expiry time, a user can still try his luck with table $\mathcal{T}_2$ and if that works out, potentially be able to restore the key in the storage locations of table $\mathcal{T}_1$ as long as the underlying scheme provides this functionality and allows to reset specific entries (as, e. g., DNS-based schemes do).

We note that we can have as many tables and hence puzzles with different difficulty levels as desired. The number of puzzles should depend on how much this affects setup time of the object and the size of the final data object. If the underlying scheme takes long time in distributing a key, it might be not be desirable to have multiple puzzles. For our evaluation we chose two tables, one guarded by an easier and one by a harder puzzle. The process of interaction with the puzzles and data retrieval are demonstrated in Figure 6.

Opposed to other uses of cryptographic puzzles, there is no need to verify the solutions to our puzzles before someone gets access to the underlying data. Incorrect solutions would only mean that the required work has not been done, leading to incorrect retrieval information about the key.

*Key Refreshment / Lifetime Extension:* The following considerations apply to schemes that enable flexible lifetime extensions. The more difficult puzzle $\mathcal{P}_2$ will have to be solved less frequently or not at all. Thus key access and refresh operations for the key stored in $\mathcal{T}_2$ will occur rarely or not at all. So we need a mechanism that

allows to refresh the key bits of table $\mathcal{T}_2$ when a refresh operation for the key of the first table $\mathcal{T}_1$ is happening—*without having to solve* $\mathcal{P}_2$.

**Mechanism:** The exact mechanism to enable the lifetime extension for the key encoded by table $\mathcal{T}_2$ depends on the type of representation of the key bits $K_i$ in the distributed storage. We here provide the idea for Neuralyzer [28] (as described in Sec. 2.4): By encrypting the tables with puzzles, we are trying to hide from an interfering attacker (who wants to delete lots of data) the table rows that represent a key bit of 0. The reason is that this attacker can delete the object by picking resolver-domain pairs that represent 0 key bits and sending recursive queries to the servers for those domains. So without solving puzzle $\mathcal{P}_2$, the attacker will not know the resolver-domain pairs representing 0 bits and hence will not be able to destroy the key in the distributed storage prematurely by sending a recursive request.

However, picking pairs that represent key bit 1 and sending recursive requests to the servers about the domains would also have an opposite effect as it will essentially extend the lifetime of the object. Due to asymmetry of 0 and 1 key bits in the scheme of Neuralyzer [28] as well as EphPub [3], 1 key bits cannot be deleted prematurely but will only be removed by the natural replacements in DNS resolver caches. That means, we can make available a copy of the addresses of the resolver-domain pairs that represent key bits 1 from the table $\mathcal{T}_2$ without requiring a user to decrypt $\mathcal{T}_2$ as they cannot be used to delete the object.

Thus, our process is as follows: Whenever we refresh a DNS entry of the easier table $\mathcal{T}_1$ (using the 1 key bit addresses in $\mathcal{T}_1$) we do the same for $\mathcal{T}_2$ without ever solving the harder puzzle $\mathcal{P}_2$ (using the 1 key bit addresses in $\mathcal{T}_2$ that we are making available now by keeping them unencrypted). So in short, additional to the encrypted table $\mathcal{T}_2$, we keep a *copy* of the unencrypted information about the resolver-domain pairs in the second table representing the key bit 1 (and only bit 1!).

**Resistance to attacks:** As far as attacks during the lifetime of an object are concerned, we note that this does not reveal exactly which bits are 1 (and hence the key) because $\mathcal{T}_2$ that captures the ordering of the 1s and 0s is encrypted. The only information that the unencrypted pairs give away is the total number of key bits that represent 1. Arguably, it is much more efficient to solve the easier puzzle $\mathcal{P}_1$ (or even the harder one) to get the actual key than to brute force the key using a reduced key space as we now know the total number of 1s the key contains.

However, if we were to leave these resolver-domain pairs unencrypted, then after the object has expired the number of 1s the key $K$ consists of would become common knowledge to an attacker. So the object would indeed become less secure because of the reduced key space. But actually if the attacker then solves $\mathcal{P}_2$ to get $\mathcal{T}_2$, it can map the unencrypted pairs to $\mathcal{T}_2$ and reconstruct the key without ever querying, even after the object has expired. That is why we encrypt this information along with the actual data content. Now whenever an easier puzzle is solved, the solver gets the key to decrypt the data content as long as the object still has not yet expired. He or she decrypts the data and along with it the 1 key bit information which is then used to extend the lifetime of the addresses of the second table $\mathcal{T}_2$ (and hence the key).

One final remark here is that an attacker with different intentions like extending the lifetime of a multitude of different objects would not be deterred by this too much as compared to an attacker who targets deletion. But this attacker would still have to solve all the easier puzzles.

## 4.3 Security Analysis

Before analyzing the security of our proposal with respect to the security goals defined in Section 2.3, we shortly discuss how it meets the functional goal defined in same section.

**Functional Goal:** Our timelock puzzle system is implemented on top of the underlying scheme used to provide digital forgetting. We encrypt the information required to retrieve the encryption key with a timelock puzzle. Anyone with access to the (encrypted) data object who can solve the (first) puzzle before the expiration time $t_e^x$, can then follow the underlying scheme to retrieve the key. If $c_1$ is the time used to solve the (first) puzzle, this provides access to specific data objects until $t_e^x - c_1$ for all parties, including curious-but-non-interfering attackers and interfering attackers. Our second puzzle is another way to get the same key using the same scheme. The data is retrievable as long as users can solve the second puzzle in time $c_2$ before $t_e^x - c_2$. We note that this reduces the data access time by a maximum of $c_1 + c_2$ (if Puzzle 2 has to be leveraged to recover the key), but argue that this time is significantly shorter than typical data lifetimes and does thus fulfill the functional requirement.

Regarding the security of our proposal, we note that it becomes harder to delete an object (specifically its key $K$) due to the presence of two puzzles. It is also hard to collect large amounts of data because a cryptographic puzzle will have to be solved in all cases. More specifically, we relate our proposal back to the three security goals:

**Security Goal 1:** After $t_e^x$, if the underlying scheme provides no further access to the data, we argue that Security Goal 1 is achieved since our system operates on top of the underlying digital forgetting scheme. In other words, we are only adding a delay before key retrieval and the puzzle does not contain information on the key bits, but only references to their storage locations. Hence, after expiry, successful recoverability of the encryption key and knowledge of the cryptographic puzzles cannot lead to a successful access to data content.

**Security Goal 2:** Before $t_e^x$, a curious-but-non-interfering attacker can save a data object for the purposes of malicious use in the future. The timelock puzzles essentially guarantee that given a machine of certain processing power (with capabilities for specific numbers of squarings modulo $n$ per second), the puzzle creator can set the time needed to solve the puzzle. Since the puzzle is sequential in nature, a solver cannot break it into sub tasks and do them in parallel. So by using the puzzle we make sure that an attacker spends at least a few seconds working on a single data object before saving it or its key. We thus limit the number of successful attacks an attacker can do per day based on the number of used machines and their processing power. While it is true that we cannot make an attacker spend too much time on a data object given the nature of our data that is meant to be consumed by normal users as well, we will discuss in Sections 5 and 6 how costly we still make these attacks for an

attacker. Hence, although we cannot fully prevent curious-but-non-interfering attacks, we make taking regular, complete snapshots much harder.

**Security Goal 3:** Before $t_e^x$, our first puzzle targets Security Goal 2. Given the inherent difference in attacks (unfocused data collection vs. targeted interference), we make it even more costly for an interfering attacker to delete a data object. More specifically, we decouple the time for curious-but-non-interfering attacks from interfering attacks by distributing the same encryption key twice. If an attacker manages to delete the key from the first location successfully, the data object has still not been deleted. Instead, the attacker must solve a more time-consuming second puzzle and delete the backup for an effective attack (note that solving two puzzles can be done in parallel, but the first puzzle will add only a few more seconds to the time needed by an attacker per object if done in a sequential manner). In other words, we make a single machine spend more time deleting an object than saving it. Naturally, doing this on a large scale for the interfering attacker is even harder to achieve than a curious-non-interfering attack against Security Goal 2 (discussed further in Sections 5 and 6).

# 5 EVALUATION

We have implemented a prototype of our scheme based on Neuralyzer [28] (for key aspects of [28] see Sec. 2.4) using DNS cache entries as its public infrastructure. It uses interest in the object as a heuristic to determine when the object should become inaccessible.

## 5.1 Implementation Details

We obtained Neuralyzer's source code from the authors and integrated our proposed scheme with it. All of our code was written in Python and it relies primarily on Python's Crypto library. We also used PyDNS (version 2.34) in our code which provides a module to perform DNS queries from python applications. We use 128-bit keys to encrypt our objects. Each bit is encoded in 8 resolver-domain pairs for redundancy like in the original scheme. A key bit is interpreted as '1' if at least 4 out of 8 of these servers return true on a non-recursive request about their respective domain where 'true' means that the domain is still in the server's cache.

To get the key to decrypt the data, one needs to get access to a table ($\mathcal{T}_1$) containing these $128 \cdot 8$ (1024) resolver-domain pairs and to get this information, a puzzle ($\mathcal{P}_1$) needs to be solved. Only those resolver-domain pairs are used where the time to live in the cache is at least two hours to make practical sense. We encode the same key in 1024 other different pairs ($\mathcal{T}_2$) and then restrict access to it by a harder puzzle ($\mathcal{P}_2$). When an object is decrypted, all the key bits representing one-key bit in both the tables are refreshed as the act of decryption shows that there is still some interest in the data. For each one-key bit, we check whether at least five of the pairs still respond true to a non-recursive request. If the number is less than five we take a pair at random and send a recursive request to the server about the pair's domain so that it is put inside the server's cache. We also check other criteria set by the original Neuralyzer code such as if the median of the time to live values of these pair is less than a pre-configured value, we must refresh.

As discussed in Section 4.2, we also need to store a copy of all the resolver-domain pairs that represent one key bit in $\mathcal{T}_2$ separately. We encrypt this copy along with the data. After solving $\mathcal{P}_1$, the user is able to decrypt $\mathcal{T}_1$ and get the decryption key for the data, which means he will be able to get access to this copy as well and will be able to refresh $\mathcal{T}_2$ without even solving $\mathcal{P}_2$. Both tables are encrypted with separate keys ($E_1$, $E_2$). The keys that encrypt the tables are modified as described in Section 3.1.1 such that these modified keys together with the information required to correct them ($\alpha$, number of times ($t$) that $\alpha$ needs to be squared, and $n$) make up the puzzles as part of the EDO (Ephemeral Data Object, App. 2.4). We set $n$ to be a product of two 1024-bit random prime numbers and $\alpha$ to be a random 512-bit number.

We ran all our experiments on an Intel Core i7-6600U laptop (4M Cache, 2.60GHz). The squarings per modulo $n$ our machine could do were 58 000. So setting $t$ to a value of 58 000 gave us a puzzle that could be solved in 1 second, $2 \cdot 58\,000$ gave us a puzzle that could be solved in 2 seconds and so on. What we call a harder puzzle is one which is set to be solved in more than 6 seconds.

We created about 350 different random EDOs whose original sizes are either 100 kB, 500 kB, 1 MB, 5 MB, or 10 MB. In our experiments, the difficulty of the easier puzzle $\mathcal{P}_1$, i. e., the time needed to solve it, is set from minimum 1 s to maximum 6 s across these objects. The difficulty of the harder puzzle $\mathcal{P}_2$ was set to times between 8 s to 1 min. We decrypted all the objects with success multiple times over a period of 8 hours. We made sure that the harder puzzle works like it is meant to by manually making the table locked by the easier puzzle invalid.

## 5.2 Results

Our experiments were divided into encryption and decryption phases.

During the **encryption phase**, the important aspects worth considering were *i*) time taken to create an EDO and *ii*) size of an EDO. Figure 7 shows how creating the puzzles takes almost negligible time compared to the time spent in finding the usable resolver-domain pairs to encode a key-bit. Usable pairs are those where the domain is not yet in the server's cache and the time-to-live (TTL) field takes a value that is appropriate for the type of content we are dealing with. For example, for a blog we might want greater TTL values compared to a tweet. So the majority of the time is spent on this process, which is costly in terms of communication. But this work of finding a pair can be done in advance, i. e., there can be a precomputed list of suitable pairs that is updated by a process running in the background that adds more pairs to this list periodically and removes if any of the pairs become unusable. Given that we have this list now, we will then just need to communicate with 2048 ($2 \cdot 128 \cdot 8$) DNS servers as we have two tables, 128-bit keys, and and a redundancy of 8 for each key bit. This will reduce communication cost during EDO creation making it faster for a user to create and upload an EDO.

The results also show that the file size has no significant bearing on the amount of time spent on creating an EDO because almost all the time is spent on the DNS communication and creating the tables and this process is in fact file independent. The average time (one time cost to create an object) was approximately 70 s if we do not process resolver-domain pairs in advance. We also see how the

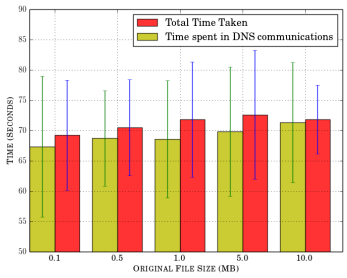


**Figure 7: Total time spent in creating an EDO vs. Time spent on communicating with DNS servers**

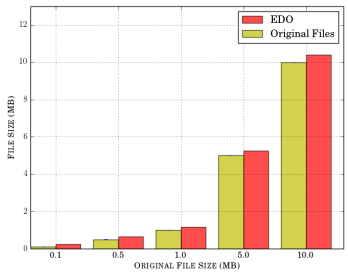


**Figure 8: Original File Sizes vs. EDO Sizes**

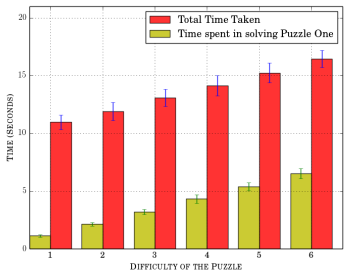


**Figure 9: Time taken to decrypt an EDO. Difficulty of the puzzle represents the expected time needed to solve it.**

work other than the communication just takes a few seconds on average.

Figure 8 depicts how the size of an EDO exceeds the actual file size. This increase starts to become negligible as the original file sizes increase. This makes sense because in every EDO, regardless of the original file size, we only add limited data such as variables for both puzzles, encrypted tables, and copy of the resolver-domain pairs representing the one key-bits in $\mathcal{T}_2$. This increase in file size is almost of a constant size.

For evaluating the **decryption phase**, we are interested in the following questions:

(1) Did the puzzles get solved in expected time?
(2) How much time does refreshing the one-bits take?
(3) What was the total time taken to decrypt an EDO?

Figure 9 shows how the puzzles were solved in about the time that they were meant to be solved in. We found out that refreshing a table took around 4 to 5 seconds on average. The taller bars in Figure 9 represent the total time taken that was equal to the time it took the user to solve the easier puzzle together with the time taken to refresh two tables (8 to 10 seconds). It needs to be stressed here that refreshing a table can be done using a background process and a user can get to see the content as soon as he solves the puzzle whose difficulty (time needed to solve) we directly control.

## 6 DISCUSSION

In this section we discuss the impact of our proposal on the success of an attacker and its performance against specialized devices. For a discussion and outline of possible integration of our proposal in online social networks (OSNs) we refer to App. A.4.

**Curious-but-non-interfering Attacker:** While proposals such as Vanish [7] and Neuralyzer [28] undermine attackers who aim to access the data objects after their expiration times, they do not provide resistance against attackers that are tricking the system by actively trying to take a snapshot of the data for future use. This kind of attacker's job is made extremely tough by our addition of puzzles over the underlying mechanism, since he will need access to a large number of machines to perform the extra computation overhead. An easy puzzle such as the one used in our implementation requires 174,000 squarings for decryption. In other words, one million of such EDOs would require 174 billion squarings to be done in order to get compromised—in addition to the effort required to retrieve the data object from the underlying mechanism such as Neuralyzer. It is estimated that 300 million photos are uploaded on Facebook daily [22]. An adversary who would be interested in saving these data objects will need to do alarming 53 trillion squarings daily.

The time it takes to do the above computations to compromise the daily 300 million uploads on Facebook varies depending on the number and the processing power of the devices used. Given the large scale of computation required, the attackers might outsource the task of taking snapshot of the system to cloud computing services. To put this effort into perspective, we leverage Amazon's EC2 cloud computing service to test the resilience of our added puzzles.

Table 1 details the approximate number of required compute-optimized EC2 C4.8xlarge instances and the corresponding *daily* monetary costs for an attacker to take varied (one quarter, one half and whole) snapshots of the photos uploaded on Facebook. The numbers show that the computation costs of conducting such a vast attack will be too high to make it feasible for the attacker to save large amounts of uploaded data on regular intervals. It is important to recognize that this additional effort is required solely to solve the puzzle(s) and does not account for the time needed to retrieve the data object from the underlying key storage mechanisms. Moreover, this analysis does not take those EDOs into consideration that expire before they were saved.

**Interfering Attacker:** Another major contribution of our proposal is the resilience added to the underlying scheme against large-scale early deletion attacks. An adversary who wants to delete, say, at least a quarter of the 1.2 billion photos uploaded on Facebook in four days would have to spend about 12,000 days on this mission if he stays within the allowed limit of 20 Amazon EC2 C4 instances, given our easy puzzle takes 3 seconds and the harder one takes a minute. This means that he would need access to roughly two hundred fifty thousand compute-optimized Amazon C4 instances for deleting 300 million photos within one day. Even if we believe in the unlikely scenario where this massive number of instances exist in the first place and that the attacker manages to get access to those, this will cost him around 9 million dollars a day to compromise these data objects (as illustrated in Table 1). This cost can

| Volume | Snapshot Attack | | | Deletion Attack | | |
|---|---|---|---|---|---|---|
| | No. of Squarings | No. of Instances | Monetary Cost | No. of Squarings | No. of Instances | Monetary Cost |
| 75 million | 13 trillion | 2,840 | $108,440 | 274 trillion | 59,625 | $2,276,720 |
| 150 million | 26 trillion | 5,680 | $216,880 | 548 trillion | 119,250 | $4,553,440 |
| 300 million | 53 trillion | 11,360 | $433,770 | 1.1 quadrillion | 238,500 | $9,106,880 |

**Table 1: The number of Amazon C4 instances required and the resulting daily costs for an attacker to perform Snapshot and Deletion attacks on 25%, 50% and 100% of the daily uploaded photos on Facebook.**

increase even further if additional puzzles of harder difficulty are added to the system.

It is important to consider that if there were no puzzles in place to solve at all, only 14,500 Amazon instances or machines of similar computational abilities would help him to save and/or delete more than a billion photos in a day easily, assuming that it takes around a second to download them/send recursive queries. All of this also does not take into account the fact that this content also can expire naturally even before the adversary gets to it; either to save or delete. And with no puzzle guarding the access to these objects, the probability of an adversary getting to the object before it expires gets higher as the adversary will be going through objects on a much greater speed in this scenario.

**Attacker with Specialized Hardware:** While a higher clock speed will ensure that more squarings are done per unit time and the puzzles are solved faster than the intended time, the clock speeds have become static in the last decade owing to size of transistors reaching its limits [14]. Thus, the performance improvements today are achieved by the use of multi-core processors and parallel computing practices instead. These techniques do not yield significant improvement on our scheme as the repeated squaring algorithm used in our proposal is essentially sequential [20]. However, the well-funded attackers can make use of specialized hardware such as FPGAs and the latest generations of GPUs, equipped with support for integer operations, in order to achieve speed up. Some researchers have shown that off-the-shelf hardware could be used to accelerate the public key cryptosystems such as RSA scheme by roughly 4 times and this could in turn be used on these puzzles as well [9]. Referencing the above calculation, it will still take an adversary to use roughly 50,000 of these specialized hardware devices to compromise only one quarter of the billion photos uploaded on the Facebook. So, the expected time improvement for such an attacker will still be in the order of a small constant number and the added puzzles will continue to limit the scale of disruption or copying attacks.

**Pre-computations:** It could be said that an adversary can solve a puzzle and start saving the solution. We argue that it will be very rare that such a strategy would benefit him. Again, let us take Facebook as an example. The number of objects uploaded in a matter of days on Facebook is in order of billions but the number of values of $\alpha$ in the puzzle is an exponentially larger number. Not to mention $\alpha$ alone does not determine the solution, but it is determined by the combination of $\alpha$, $n$ and $t$. So saving solutions would not make a significant difference at all. And we can increase the length of these numbers as well, leading to a much bigger space from which puzzles can be generated. Creating such a storage by a single adversary with a finite number of machines appears prohibitive. If the attacker were to decrypt these billions and billions of objects uploaded in a year across major social networks and other platforms and assuming all the puzzles on these objects are supposed to be solved in 5 seconds, for a single machine this task will take more than a million years.

## 7 CONCLUSION

In this paper, we outlined and investigated a general use case of cryptographic puzzles to support the concept of digital forgetting. The pervasiveness and ubiquity of digital data creates a desideratum for solutions that enable transientness and ephemerality of person-related information, data, and media. The attacker model here differs from classical attackers since we are trying to protect data from access and deletion during its lifetime when the data is supposed to be publicly available. The approach we propose introduces an asymmetry between regular users and active attackers and specifically takes into account the attacks that attempt to delete public data during their lifetime. Our proposal makes it hard for the attacker to delete large quantities of data while making sure that the proposed solutions will not adversely deteriorate user experience in a disturbing manner. Our system relies on cryptographic time-lock puzzles and deals with malicious users while ensuring the permanent deletion of data when interest in it dies down. We have implemented a prototype and evaluated it thoroughly with promising results, which makes a case for further research in this area.

## 8 ACKNOWLEDGEMENTS

This work was supported by the Center for Cyber Security at NYUAD and by the German BMBF under grant 16KIS0581.

## REFERENCES

[1] Adam Back. 2002. Hashcash – A Denial of Service Counter-Measure. (2002). http://www.hashcash.org/papers/hashcash.pdf.

[2] Dan Boneh and Richard J. Lipton. [n. d.]. A Revocable Backup System. In *Proceedings of the 6th USENIX Security Symposium, San Jose, CA, USA, July 22-25, 1996.*

[3] C. Castelluccia, E. De Cristofaro, A. Francillon, and M.-A. Kaafar. 2011. EphPub: Toward robust Ephemeral Publishing. In *19th IEEE International Conference on Network Protocols (ICNP)*. 165–175. https://doi.org/10.1109/ICNP.2011.6089048

[4] Domo. 2017. Data never sleeps 5.0. https://www.domo.com/learn/data-never-sleeps-5. (June 2017). https://www.domo.com/learn/data-never-sleeps-5

[5] Sujata Doshi, Fabian Monrose, and Aviel D. Rubin. 2006. Efficient Memory Bound Puzzles Using Pattern Databases. In *Proceedings of the 4th International Conference on Applied Cryptography and Network Security (ACNS)*. 98–113.

[6] Roxana Geambasu, Tadayoshi Kohno, Arvind Krishnamurthy, Amit Levy, Henry Levy, Paul Gardner, and Vinnie Moscaritolo. 2011. *New Directions for Self-Destructing Data Systems.* Technical Report UW-CSE-11-08-01. University of Washington.

[7] Roxana Geambasu, Tadayoshi Kohno, Amit A. Levy, and Henry M. Levy. 2009. Vanish: Increasing Data Privacy with Self-destructing Data. In *Proceedings of the 18th Conference on USENIX Security Symposium*. USENIX Association, Berkeley, CA, USA, 299–316.
[8] Peter Gutmann. 1996. Secure deletion of data from magnetic and solid-state memory. In *Proceedings of the 6th USENIX Security Symposium, Focusing on Applications of Cryptography*. USENIX Association, Berkeley, CA, USA, 77–90.
[9] Owen Harrison and John Waldron. 2009. Efficient Acceleration of Asymmetric Cryptography on Graphics Hardware. In *Proceedings of the 2Nd International Conference on Cryptology in Africa: Progress in Cryptology (AFRICACRYPT '09)*. Springer-Verlag, Berlin, Heidelberg, 350–367.
[10] Ari Juels and John G. Brainard. 1999. Client Puzzles: A Cryptographic Countermeasure Against Connection Depletion Attacks. In *Proceedings of the Network and Distributed System Security Symposium, NDSS*.
[11] Ghassan Karame and Srdjan Capkun. 2010. Low-Cost Client Puzzles Based on Modular Exponentiation. In *Proceedings of the 15th European Symposium on Research in Computer Security (ESORICS)*. 679–697.
[12] Jaeheung Lee, Sangho Yi, Junyoung Heo, Hyungbae Park, Sung Y. Shin, and Yookun Cho. 2010. An Efficient Secure Deletion Scheme for Flash File Systems. *Journal of Information Science and Engineering* 26, 1 (2010), 27–38. http://www.iis.sinica.edu.tw/page/jise/2010/201001_03.html
[13] Frank Li, Prateek Mittal, Matthew Caesar, and Nikita Borisov. 2012. SybilControl: Practical Sybil Defense with Computational Puzzles. In *Proceedings of the Seventh ACM Workshop on Scalable Trusted Computing (STC'12)*. 67–78.
[14] Pär Persson Mattsson. 2013. Why Haven't CPU Clock Speeds Increased in the Last Few Years? (Nov 2013). https://www.comsol.com/blogs/havent-cpu-clock-speeds-increased-last-years/
[15] Viktor Mayer-Schönberger. 2011. *Delete: The Virtue of Forgetting in the Digital Age*. Princeton University Press, Princeton, NJ, USA.
[16] Mainack Mondal, Johnnatan Messias, Saptarshi Ghosh, Krishna P. Gummadi, and Aniket Kate. 2016. Forgetting in Social Media: Understanding and Controlling Longitudinal Exposure of Socially Shared Data. In *Twelfth Symposium on Usable Privacy and Security, SOUPS 2016, Denver, CO, USA, June 22-24, 2016*. 287–299.
[17] Joel Reardon, David A. Basin, and Srdjan Capkun. 2013. SoK: Secure Data Deletion. In *IEEE Symposium on Security and Privacy (SP), Berkeley, CA, USA*. 301–315.
[18] Joel Reardon, Srdjan Capkun, and David A. Basin. 2012. Data Node Encrypted File System: Efficient Secure Deletion for Flash Memory. In *Proceedings of the 21th USENIX Security Symposium*. 333–348.
[19] Sirke Reimann and Markus Dürmuth. 2012. Timed revocation of user data: long expiration times from existing infrastructure. In *WPES*. 65–74.
[20] Ronald. L. Rivest, Adi Shamir, and David A. Wagner. 1996. *Time-lock Puzzles and Timed-release Crypto*. Technical Report. Cambridge, MA, USA.
[21] Esther Shein. 2013. Ephemeral Data. *Commun. ACM* 56, 9 (Sept. 2013), 20–22.
[22] Cooper Smith. 2013. Facebook Users Are Uploading 350 Million New Photos Each Day. (Sep 2013). http://www.businessinsider.com/facebook-350-million-photos-each-day-2013-9
[23] Douglas Stebila, Lakshmi Kuppusamy, Jothi Rangasamy, Colin Boyd, and Juan Gonzalez Nieto. 2011. Stronger Difficulty Notions for Client Puzzles and Denial-of-Service-Resistant Protocols. In *Proceedings of CT-RSA 2011: The Cryptographers' Track at the RSA Conference 2011*. Springer Berlin Heidelberg, 284–301.
[24] XiaoFeng Wang and Michael K. Reiter. 2004. Mitigating Bandwidth-exhaustion Attacks Using Congestion Puzzles. In *Proceedings of the 11th ACM Conference on Computer and Communications Security (CCS '04)*. 257–267.
[25] Michael Wei, Laura M. Grupp, Frederick E. Spada, and Steven Swanson. 2011. Reliably Erasing Data from Flash-based Solid State Drives. In *Proceedings of the 9th USENIX Conference on File and Stroage Technologies (FAST)*. USENIX Association, Berkeley, CA, USA, 8–8.
[26] Scott Wolchok, Owen S. Hofmann, Nadia Heninger, Edward W. Felten, J. Alex Halderman, Christopher J. Rossbach, Brent Waters, and Emmett Witchel. 2010. Defeating Vanish with Low-Cost Sybil Attacks Against Large DHTs. In *Proc. 17th Network and Distributed System Security Symposium*.
[27] Y. Wu, Z. Zhao, F. Bao, and R. H. Deng. 2015. Software Puzzle: A Countermeasure to Resource-Inflated Denial-of-Service Attacks. *IEEE Transactions on Information Forensics and Security* 10, 1 (2015), 168–177.
[28] Apostolis Zarras, Katharina Kohls, Markus Dürmuth, and Christina Pöpper. 2016. Neuralyzer: Flexible Expiration Times for the Revocation of Online Data. In *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy (CODASPY'16)*. 14–25.

# A APPENDIX

## A.1 Extension of Puzzles On Vanish

An encrypted data object in Vanish [7] is essentially a tuple consisting of a random access key $L$ which is used to generate indices where the shares of encryption key $K$ are stored, Ciphertext ($C$), Threshold ($T$, where $T$ is the percentage of key shares required for the successful reconstruction of $K$) and $N$ as the total number of shares $K$ was split into.

The ideas described in this paper can be applied to Vanish. We can have a puzzle that guards $L$. The attacker has to solve the puzzle first to get information about how to retrieve and save $K$. Note that this does not defeat the successful attack on Vanish (Unvanish) [26] which basically targets weaknesses in Vuze, the distributed hashtable that stores the shares of $K$. Neither was this our intention in the first place as our proposal does not fix the weaknesses of the underlying schemes. Possible defenses against Unvanish are listed in [6].

## A.2 Related Work

We further describe related work more broadly: *i*) general solutions for the deletion of data and *ii*) context as well as applications of cryptographic puzzles.

**Deletion of Data.** Approaches for secure data deletion have been investigated at multiple layers of abstraction [17]: from user-level approaches [8], to deletion in file systems [18] and hardware deletion techniques [25]. User behaviors wrt. deletion have been analyzed for the Facebook OSN [16]. Orthogonal to these approaches—and closer to our investigations—are techniques that focus on the secure deletion of data by securely deleting a key that encrypts that data in *offline* contexts. The encrypted data becomes infeasible to recover without knowledge of the encryption key, given that computational hardness assumptions hold for the encryption. Examples include the early revocable backup system [2] as well as the secure deletion in the YAFFS file system [12].

**Applications of Cryptographic Puzzles.** It has been suggested that computational puzzles can be used to deal with Sybil attacks [13]. Vanish was mentioned as a system that could benefit from this as it relied on a distributed hash table which could be infiltrated by sybil nodes [26]. But the suggestion was to add these puzzles to the underlying architecture, DHT in Vanish's case. In our setting, we consider a different setup where puzzles are included inside the data objects. We are not making any changes to the underlying public infrastructures and we are not trying to defend them against exploits. We are trying to solve the problem where the attackers abuse the data object itself. Also, they use a puzzle and solution verification process that is distributed in nature and do not rely on a central server or completely different third party for verification of work done. However, their scheme is still very different from ours as it deals with a system where the aim is to bar entry of dishonest nodes who will not solve puzzles. The nodes are supposed to solve puzzles periodically and send the solutions to other nodes for verification. In contrast, we propose a self-decrypting/self-verifying scheme which means that there is no communication cost.

Cryptographic puzzles have been proposed and investigated in further contexts [11, 23]. The ideas by Stebila et al. in [23] are mostly applicable to scenarios in which server and clients are communicating, which is not the case for us. They also observe that a puzzle is 'strong' if an adversary takes $n$ amount of time to solve a puzzle, for 30 puzzles the total time should not be less than $30n$. In the spirit of this observation we argue that caching results will not help the adversary to do better as discussed in Section 6. Proposals such as [11] need a verification of the solution step as an extra round of

communication, which we cannot incorporate. Their main aim was also to reduce the time for puzzle generation by exploiting a certain class of keys where the private key was much smaller than the public key, leading to less time spent on generation and verification. They did not have any control over the time spent on solving the puzzles. Our scheme on the other hand generates a puzzle fairly quickly and does not need a separate verification step.

To the best of our knowledge, the idea to use puzzles to support digital forgetting has not been investigated before. We demonstrate that they can be used for making users do work in order to access data objects that are tailor-made to be forgotten with the passage of time, which allows to solve various problems and prevent attacks for approaches that deal with digital forgetting.

## A.3 Costs for Attacks on Facebook

Table 1 details the approximate number of required compute-optimized EC2 C4.8xlarge instances and the corresponding *daily* monetary costs for an attacker to take varied (one quarter, one half and whole) snapshots of the photos uploaded on Facebook.

## A.4 Integration in OSNs

We wrote a Firefox extension for Neuralyzer that worked with textual content. By natural extension the proposed scheme in this paper can be also turned into an extension that works on emails and various social networks. If a social network limits the type of file that can be uploaded, for example it does not recognize EDO due to any factor or it only accepts files of a specific type that it can verify before uploading, a user can upload the EDO to a cloud storage and share the link on that social network instead.

Objects on a social network need to be accessed fast. We can run background processes that keep decrypting and caching objects before a user scrolls down to them to reduce the time delay. A normal user is not looking to access even a hundred thousand objects in a day. These background processes can, as the objects flow into social media timelines, decrypt and cache them in the users' devices for a sensible period of time. We do recognize that this will have some effect on the experience. But this might be for now the price that we have to pay for enabling digital forgetting. Specific studies can be conducted in the future as to how much this affects user experience on an online social network or given the option will people care enough to opt for it or can we make compromises with respect to user experience if we move from the domain of OSNs to emails etc.