FlashGhost: Data Sanitization with Privacy Protection Based on Frequent Colliding Hash Table

Yan Zhu, Shuai Yang
School of Computer and Communication Engineering
University of Science and Technology Beijing
Beijing, 100083, China
zhuyan@ustb.edu.cn, echoyyyang@gmail.com

William Cheng-Chung Chu
Department of Computer Science
Tunghai University
Taichung, Taiwan
cchu@thu.edu.tw

Rongquan Feng
School of Mathematical Sciences
Peking University
Beijing, China
fengrq@math.pku.edu.cn

***Abstract*—Today's extensive use of Internet creates huge volumes of data by users in both client and server sides. Normally users don't want to store all the data in local as well as keep archive in the server. For some unwanted data, such as trash, cache and private data, needs to be deleted periodically. Explicit deletion could be applied to the local data, while it is a troublesome job. But there is no transparency to users on the personal data stored in the server. Since we have no knowledge of whether they're cached, copied and archived by the third parties, or sold by the service provider. Our research seeks to provide an automatic data sanitization system to make data could be self-destructing. Specifically, we give data a life cycle, which would be erased automatically when at the end of its life, and the destroyed data cannot be recovered by any effort. In this paper, we present FlashGhost, which is a system that meets this challenge through a novel integration of cryptography techniques with the frequent colliding hash table. In this system, data will be unreadable and rendered unrecoverable by overwriting multiple times after its validity period has expired. Besides, the system reliability is enhanced by threshold cryptography. We also present a mathematical model and verify it by a number of experiments, which demonstrate theoretically and experimentally our system is practical to use and meet the data auto-sanitization goal described above.**

***Index Terms*—data privacy, data self-destructing, secret sharing, hash collision, data sanitization**

I. Introduction

Internet users create voluminous data every day. The Trash (or Recycle bin) is easily piled up with unwanted files if a user forgets to empty it periodically. And users are very often upset by the caches and temporary files from software, which cause insufficient disk space and poor-organized file system. Those files are continually accumulated during using computer, and they're scattered everywhere in the computer library so that makes it difficult to clean them. As another example, considering the case of live chat, messages like home address, passport details, or credit card numbers, which should be deleted after being read. Since keeping them in the server exists a potential privacy risk, especially if the service provider is untrusted.

Therefore, we need a system that allows data to be automatically destroyed by itself, after a certain period. That not only does remove the risk of future data breach in the aspect of privacy protection, but also is helpful to keep a neat and well-organized computer environment for users.

In addition, reasons behind this lie in users' extensive reliance on web service. According to the Internet World Stats [1], up to 31st Dec 2017, the global Internet penetration is 54.4 percent. More than 4.15 billion people around the world use the Internet today, and over 460 million Internet users were added in 2017 [2]. This tells us parallelly that a huge volume of data is being created every day. However, the rapid growth of data on the Internet does not bring more really useful information, but generates large portions of redundant outdated data for users. Even worsely, it provides convenience for some organizations to retrieve information, and analyze it with the big data techniques to make profits. On the other side, data breaches happen daily. It seems as though not a day goes by without a headline reporting that some organizations has experienced a data breach or they invade users' privacy. Users also fear to have their online information tracked by others. A survey taken in 2015 found a dramatic 97% of matrimonial attorneys have seen an increase in divorce evidence being taken from mobile apps (e.g., Facebook and Twitter) during the past three years [3]. Besides being used as law evidence, some cybercriminals make use of personal information in Internet fraud or other malicious purposes. Recently, Facebook came under harsh criticism from users because of the data scandal what the Observer described as unprecedented data harvesting. That a political consulting firm Cambridge Analytica hired by Donald Trump's presidential campaign, harvested private information from the Facebook profiles of more than 50 million users without their permission to influence voter opinion [4]. These all undoubtedly raise the users' great concern about the data privacy.

In 2011, three students from Stanford University created an ephemeral messaging mobile application, called Snapchat. Soon after, it was evolving into one of the most popular social media applications. One of the principal concepts of Snapchat is that messages are only available for a short time before they become inaccessible. We call such a message an ephemeral message. The solution of the problem tackled in this paper is just take advantage of this idea, data self-destruction.

We target the goal of a system with automatic data saniti-

2474-2473/19/$31.00 ©2019 IEEE
DOI 10.1109/SCC.2019.00026

zation functionality, creating data that self-destructs or disappears after it is no longer useful without any explicit action by the users.

Numerous applications could benefit from such self-destructing data. Such as Trash (a temporary storage for deleted files of Mac) and temporary files directory, could reduce a lot of maintenance to users, leaving a clean computer environment, by applying this system. As another example, considering the case of online communication. Imagine that Alice wants to have live chat with Bob. Concerning the discussion involves privacy, they don't want to leave a history record both in local and server. Therefore, they hope the messages could be destroyed automatically after a limited certain time rather than by manual delete action. In the normal instant messaging application, message is not self-destructing, and there is no guarantee that the message is deleted in the server while it is deleted on application by user. In order to achieve this goal, the natural solution to this situation is simply encrypting the message. First, Alice and Bob choose an encryption scheme and exchange the key by side channel (e.g., a phone call, or by SMS). Then they encrypt the message in local and send the encrypted message to each other via an instant messaging app. After the conversation is over, they destroy the key to make the messages unreadable. In a sense, this solution provides a way to achieve data self-destruction. Ephemerizer [5] just takes advantage of this idea to create, advertise, and destroy keys to keep data for a finite time. Other schemes, like FADE [6] and CloudDocs [7] work in a similar manner to achieve secure deletion.

However, this approach bears limitations in implementation. For normal users, they have to be equipped with a certain cryptography knowledge to choose an appropriate encryption scheme. For a big system like Ephemerizer, it must be a trusted and reliable third party. Furthermore, if using symmetric encryption scheme, there is a big problem in the aspect of key's generation, distribution and management. Additionally, because of the limitation of identity certificate distribution, this case cannot employ public key scheme as well. Even though the above difficulties can be solved, the way to destroy the key and left encrypted data only sacrifice the readability of data, while don't realize erasure on the storage device. More or less, it exists potential risk if we assume the adversary has strong enough computing ability to decrypt data.

Related Work. Geambasu et al. implemented a prototype system, called Vanish [8], which is in the context of cloud computing and integrates cryptography techniques with global-scale, P2P, distributed hash tables (DHTs). The following is a brief summary of its workflow. Firstly, Vanish encrypts data D with a random key K to get ciphertext C, then using threshold secret sharing [9] to split K into N pieces, which will be derived into the DHT. Next, the ciphertext C and the parameters to retrieve K are encapsulated into a data object called VDO. Finally, receiver reconstructs K with VDO and decrypts C to obtain D before its expiration. The key insight of Vanish is to leverage the availability of DHT, which has a specific timeout to stored data. Based on this work, many research optimized Vanish and designed new models, such as SafeVanish [10] and SSDD [11].

However, there are some deficiencies choosing DHTs as storage systems for Vanish. On the one hand, the ciphertext is still stored in VDO. It doesn't remove thoroughly the risk of reconstruction (even though it's very difficult). On the other hand, even if the constant churn in the DHT is an advantage to Vanish because of the self-cleansing property, but it also makes the access to shares and the expiration of key unstable and uncertain. Furthermore, a research confirmed that Vanish system is vulnerable to Sybil attacks [12]. To address this problem, SeDas system [13] is proposed, which integrates asymmetric key techniques with active storage techniques. And the other optimized schemes only improve the cryptography scheme, not to the negative effects of the P2P network.

Our Goals. Hence, it is still necessary to find an effective solution to enhance the security and reliability of data self-destruction. We need a system, which not only encrypts data but also creates a self-destructing mechanism. Motivated by that, using live chat as an example application, five main goals of our system are listed below:

1. *Absolute self-destruction*, completely destroy data residing on a hard disk drive or other digital media by overwriting it many times;
2. *Efficiency*, fast encryption/decryption without needing to modify any statues of data once it is created;
3. *Irreversibility*, the adversary cannot retroactively obtain the original data even get the chat history before timeout;
4. *User-friendliness*, it should be no any explicit action by the users except for the timeout setting over data;
5. *Independence*, without relying on the introduction of any third party.

A system achieving these goals would be broadly applicable to the various data types and digital contents, not only the text, such as files, pictures and emails, etc. The introduction of self-destructing data could potentially enhance the privacy protection.

Our Approach. We've previously noted some research which aim to implement self-destructing mechanism. However, they don't achieve absolute data sanitization because of leaving the encrypted data in the server while not erasing the data. Also, these techniques cannot accurately control message self-destruction cycle. To render data unrecoverable and achieve permanent data erasure, we can simply overwrite data many times. Motivated by this idea, our approach is to leverage the hash collision and secret sharing scheme. Specifically, we first split data into a group of encrypted shares, then assigning them random indexes to store in a designed frequent colliding hash table. When a used index mapped, a hash collision occurs, which means the new data would overwrite the old data over that node in hash table. Our choice of secret sharing stems from two properties that make it attractive for our challenging goals. First, it's a simple but strong encryption scheme. Second, it's a threshold scheme. Take $(k,\ n)$-threshold as an example, at least k out n shares can decrypt the ciphertext, which provides the durability of data lifetime by increasing redundancy for the loss of shares.

Our Contributions. In this paper, we present a data sanitization system in the aspect of privacy protection, called FlashGhost, to achieve secure self-destruction of data. Looking

ahead and briefly considering other tempting approaches for creating self-destructing data, the main contributions of this work are summarized as follows:

- We introduce a novel data self-destructing model that combines cryptography with the frequent colliding hash table (FCHT). This model can split data into shares then store them randomly into FCHT. Shares will be overwritten automatically within a certain time after its expiration. This approach renders the data disappear from the hard drive, which cannot be recovered by any means.
- We introduce mathematical models to theoretically evaluate the performance of secure data self-destructing, and conduct a set of case studies by varying affecting factors to system performance.

The proposed system achieves the goal of secure data self-destructing, and, in a sense, it provides users with an approach to control their personal data on the Internet, whether the Web service providers are trusted or not.

Organization. The rest of the paper is organized as follows. System model, the overview of schema and threat models are introduced in Section II. The construction of FlashGhost is provided in Section III. We propose a mathematical model and case studies in Section IV. Performance analyses are presented in Section V. We conclude this paper in Section VI.

II. System Architecture

We present a new system model, called FlashGhost, providing users with secure data self-destruction service. Our model can be constructed on the local file system, e.g., trash can, that offers periodic data auto-cleaning, or on existing ECSs, e.g., Snapchat, that allows user send ephemeral messages. To support our target application (live chat with self-destructing messages, see Section I), we propose a system model (see Fig. 1), which contains following entities:

- Clients
- Instant messaging application
- Data self-destructing system, FlashGhost
- Web service provider

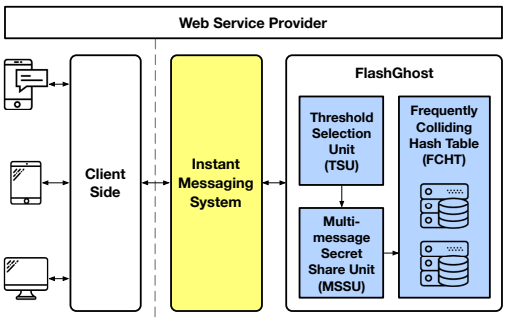


Fig. 1: Block diagram of FlashGhost.

Fig. 1 illustrates the basic system architecture. In this figure, we introduce three new modules, Threshold Selection Unit (TSU), Multi-message Secret Sharing Unit (MSSU) and Frequently Colliding Hash Table (FCHT), into the existing instant messaging system (IMS). The TSU is responsible for threshold generation, which depends on the validity period of a message. The MSSU, is a cryptographic module splitting the message into shares with the threshold scheme created by TSU. The last module, FCHT, is storage module, where stores the shares.

Our design objective is to empower users with greater control over their private data and provide real data self-destruct functionality. The following three points shall be noted: first, all the data stored in this system has a limited life. Users could set a timer to each message, and the timer is started as the message being sent, not as being read; second, it is possible that a message is expired before its lifetime; last but not least, this system must under the protection of TLS/SSL protocol for guaranteeing secure communication, removing the risk of Sniffing attacks and etc.

A. An Overview of Schema

The key insight behind our approach to achieve secure data self-destructing is to leverage the merits of hash collision. A hash table uses a hash function to compute an index into an array of buckets or slots. A collision occurs when two different inputs produce the same output. With this property, we can make the message unrecoverable by overwriting multiple times, all by automatically. Our choice of $(k,\ n)$-threshold secret sharing scheme is for security and accessibility reasons, which means any k out of n shares may be used to recover the secret. It not only encrypts data, limiting adversary's ability to obtain a sufficient number of shares, which must be $\geq k$, but also allows data to be accessible before the timeout, otherwise the loss of a single share would cause the loss of the message.

Considering the live chat example mentioned above (see Section I), we can summarize the work flow of our system as follows:

1. Alice and Bob log into FlashGhost after verified. The system creating a secure communication connection between each other.
2. Alice inputs the message M, then set a view count and a validity period.
3. After sending, a keyword K and a $(k,\ n)$-threshold scheme depending on the user's setting would be created by TSU.
4. Then M is encrypted and split into n pieces by MSSU.
5. The system sprinkles these n pieces into the FCHT (indexes are up to K), and then send the keyword K to the receiver.
6. Bob clicks the keyword received. FlashGhost would make a verification in the background, if the message is valid, the decrypted text would be presented to Bob; if not, an invalid hint would be popped up.

B. Threat Model

We now consider potential adversaries against FlashGhost. The core of our system is to achieve complete destruction of data after being invalid, not unreadability of message in a cryptographic sense. Therefore, we focus threat model and subsequent analysis on attackers who wish to compromise data privacy. The threat model can be divided into two categories:

1. Attackers pry into personal information by accessing encrypted data stored in the server before the message expires.
2. Attackers attempt to obtain evidence by using data recovery utilities after the message expires.

Firstly it's useless to intercept data under the protection of TSL/SSL protocol, and the message must be reconstructed with sufficient shares (secret sharing); secondly, we assume that the attacker's ability is strong enough to obtain all the encrypted data from server. But it's extremely difficult to decrypt the data, if the attacker does not know the information of shares. Besides, collisions occurring all the time, which restricts attacker's ability greatly.

The latter aspect of the threat model is straightforward: once the message becomes invalid, it would be overwritten multiple times within a short time. According to the DoD 5220.22-M data sanitization method [14], 7-pass erasure using random data will do a pretty complete work to prevent reconstruction. Therefore, it is impossible for anyone to recover data which has been destructed physically.

In summary, our FlashGhost model has following advantages in comparison with the other model: 1) Destroy automatically the expired message via physical approach; 2) Restrict the ability of attackers by threshold cryptography and dynamic hash table.

III. System Design

In this section, we present our method for the design of FlashGhost system, based on the system model proposed in Section II. By using this system, the data will be erased automatically at the end of its validity period, and adversaries could not extract traces of destroyed data so as to eliminate the user's concern about privacy leakage.

We say that data is secure self-destruction if it satisfies the following definition:

*Definition 1 (**Secure Data Destruction**): Secure Data Destruction is a secure technique for the removal of sensitive data from storage pools in such a way that there is an assurance that the data may not be reconstructed by using normal data recovery utilities.*

There are several practical methods for securely deleting data from a certain medium. Governments and industries have created some standards for software-based overwriting that removes data. A key factor in meeting these standards is the number of times the data is overwritten. A survey [15] for addressing secure deletion problem has explored various approaches in detail. Without loss of generality, we assume that our system is built on traditional magnetic disks. To destroy all electronic data residing on a magnetized medium, data overwriting, by using zeros and ones to overwrite data onto all sectors of the device, could be considered as assured deletion. Hence, we apply this method to realize secure data destruction. Based on Definition 1, we could define the self-destructing data in the following:

*Definition 2 (**Self-destructing Data**): Data is said to be self-destructing if the data can be automatically destroyed in the process of writing and reading, rather than the specific deletion command, at the end of its validity period.*

In this design, the ephemeral messages are the self-destructing data. With these two definitions, we now turn our attention to the framework of the system.

A. Access Policy and Storage Strategies

Most data self-destructing schemes are only time-specific. Such as Vanish depends on the lifetime of a node in the DHT, and schemes like KP-TSABE [16] could support user-defined time intervals. Besides, SecureSnaps [17] uses counting control to support forward secrecy. In our data self-destructing system, we use two indicators, duration and count, to verify whether a message is "active". The following two conditions are used to define a simple control over message activity:

1) *Time control*: refers to a defined period of time to the message, namely its validity period, represented as $[t_a, t_b]$;
2) *Counting control*: refers to the view count for the number of times the message can be viewed by receivers, represented as c.

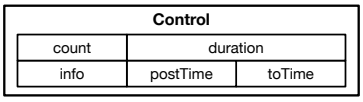


Fig. 2: An example block structure of access policy.

We say that a message is "active" if and only if two conditions are satisfied simultaneously, that is, $c > 0$ and $t_a \leqslant t_n \leqslant t_b$, otherwise, it is "invalid", where t_n is the current time. And the precondition of destructing a message is that one of the indicators exceeds its limitation. Once message become invalid, it would be tagged as unreadable to users, waiting to be overwritten by new data. An example structure of access policy is illustrated as Fig. 2. In this figure, $count$ is the available access times of a message which will be updated after each query; $duration$ represents the validity period; $postTime$ and $toTime$ are equal to t_a and t_b, respectively, and $info$ contains some extra information, such as the public parameters of secret sharing scheme.

With regard to data storage in this system, we just save data directly in Frequently Colliding Hash Table.

B. Threshold Selection Strategy

Threshold is a concept in secret sharing. Considering $(k,\ n)$-threshold scheme, k is the threshold and n is the total number of participants, which any group of k or more participants can together reconstruct the secret but no group of fewer than k participants can. While the previous subsection gives an overview of the access policy, our threshold selection strategy is based on it. Messages with different access policies (e.g. messages can be read three times and ten times within 30 seconds) require different threshold schemes. Therefore, how to generate a suitable threshold scheme for each message is the key discussion in this part. Three possible strategies are provided as follows:

*Strategy 1: **Absolute threshold schemes.*** We use c and d to denote message view count and validity period, respectively,

and let c' and d' be the default values. In this strategy, we divide all access policies into five categories by introducing c' and d', which are listed as follows:

1) View count $c > c'$, validity period $d < d'$,
2) View count $c > c'$, validity period $d > d'$,
3) View count $c < c'$, validity period $d < d'$,
4) View count $c < c'$, validity period $d > d'$,
5) View count $c = c'$, validity period $d = d'$.

Absolute threshold schemes mean we only use several predefined schemes for all cases. The specific value of c' and d' are determined by the system parameters (i.e. the capacity of hash table and the insert frequency of new data), or the application requirement (i.e. the minimum and maximum lifetime of the message).

It's the simplest way to generate a threshold scheme according to these five cases. Likewise, we can enumerate more possible cases of access policies and get more accurate threshold schemes.

Strategy 2: ***Weight calculation***.

Suppose k and n can be calculated by introducing coefficients $\alpha_1, \beta_1, \alpha_2$ and β_2, as shown in Equation 1.

$$\begin{aligned} k &= \lceil \alpha_1 c + \beta_1 d \rceil, \\ n &= \lceil \alpha_2 c + \beta_2 d \rceil. \end{aligned} \tag{1}$$

To determine these four coefficients, we could analyze the activeness of message under various threshold schemes. The theoretical values of message's lifetime can be calculated by the mathematical model (see Section IV).

Strategy 3: ***Only consider the validity period.***

Since invalid data would be overwritten within a short time after expiring, without the loss of readability, we can simply let validity period d be the decisive factor to generate threshold scheme. One of the benefits of this strategy is simple logic and high efficiency by which reducing parameters to only single one. Furthermore, it ensures the readability of messages.

These three strategies are only for reference. In the actual application, finding the optimal strategy requires to experimental tests and combines the theoretical analysis with the results for further discussions.

C. Multi-message Secret Sharing

A key contribution of our work is to leverage Shamir's secret sharing scheme. We use secret sharing to split message into pieces. On the one hand, it limits adversaries' ability to get message, since only obtaining a sufficient number of shares can reconstruct the message. On the other hand, we rely on its property of threshold ratio, which allows us to guarantee the accessibility of message by adding shares.

Suppose we use $(k,\ n)$-threshold scheme to share the message M. The Shamir's scheme is based on polynomial interpolation. Firstly, it picks a prime p which is bigger than both M and n, then creates a polynomial $f(x)$ of degree $k-1$ with M as the first coefficient a_0 and the remaining coefficients $a_1, a_2, \cdots, a_{k-1}$ are randomly chosen from a uniform distribution over the integers in $[0, p)$ (p is a prime which is bigger than both M and n). Next, find n distinct points on the curve as shares and give them to n participants, respectively. Only at least k out of the n shares could reconstruct M. Using Lagrange Interpolating Polynomial (see Equation 3) could reconstruct the message quickly and efficiently.

Our approach to secret sharing, called multi-message secret sharing (MSS), is slightly different, instead of taking M as the first coefficient, we split M into k substrings with equal length, as the coefficients of polynomial $a_0, a_1, \cdots, a_{k-1}$ (see Equation 2a). Thus, the reconstruction needs to compute all coefficients rather than only the first in Shamir's scheme. Finally, concatenating all coefficients and removing padding zeros to get M (see Equation 2b).

$$f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{k-1} x^{k-1} \tag{2a}$$

$$M = a_0 || a_1 || a_2 || \cdots || a_{k-1} \tag{2b}$$

The concrete description of our secret sharing scheme is presented as follows:

Initialization preparation: Given a $(k,\ n)$-threshold scheme, we pick a prime p firstly. Then we split message M into l substrings with equal length (padding zeros to last part if it is necessary) as the coefficients of polynomial $a_0, a_1, \cdots, a_{l-1}$, and $l = \lceil \|M\| / \|p\| \rceil < k-1$ where $\|M\|$ denotes the length of M and $\|p\| = \lfloor \log_2 p \rfloor + 1$. Considering enhance the reliability of our scheme, there must be at least one random coefficient. Therefore, the remaining coefficients $a_l, \cdots, a_{k-1}$ are chosen randomly.

Shares generation: Let x_i $(1 \leqslant i \leqslant n)$ be n distinct integers in a finite field $\mathbb{Z}_p$. For each $1 \leqslant i \leqslant n$, compute $y_i = f(x_i)$, where $f(x) = \sum_{j=0}^{k-1} a_j x^j \mod p$. Algorithm 1 outlines this step, in which $a[0], \cdots, a[k-1]$ are equivalent to $a_0, \cdots, a_{k-1}$. Then storing all (x_i, y_i) in our container, Frequently Colliding Hash Table.

Algorithm 1 Secret Shares Generation

Input: M: the message, k: the threshold, n: the total number of shares, p: the prime

1: Split M into l pieces as $a[0], \cdots, a[l-1]$ of length that not longer than $\|p\|$
2: Choose $k - l$ random coefficients $a[l], \cdots, a[k-1]$
3: **for** $x = 1$ to n **do**
4: $\quad shares[x] = a[0] + a[1]x + \cdots + a[k-1]x^{k-1} \mod p$
5: **end for**
6: **return** $shares$

Reconstruction: We can find the coefficients $a_0, a_1, \cdots, a_{k-1}$ of the polynomial using Lagrange interpolation

$$f(x) = \sum_{j=1}^{k} \left(y_{i_j} \prod_{1 \leqslant t \leqslant k, k \neq j} \frac{x - x_{i_t}}{x_{i_j} - x_{i_t}} \right) \mod p,$$

which is equivalent to solving the system of linear equations $V \cdot \vec{a} = \vec{y}$ for $\vec{a} = (a_0, a_1, \cdots, a_{k-1}), \vec{y} = (y_1, y_2, \cdots, y_k)$ with a $k \times k$ Vandermonde matrix V. The non-vanishing of the Vandermonde determinant for distinct points $x_1, x_2, \cdots, x_n$

shows that, for distinct k points (x_i, y_i), the polynomial interpolation problem is solvable with unique solution, i.e.,

$$\begin{pmatrix} 1 & x_1 & \cdots & x_1^{k-1} \\ 1 & x_2 & \cdots & x_2^{k-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_k & \cdots & x_k^{k-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{k-1} \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_k \end{pmatrix}.$$

There are two approaches to compute the unique solution of $a_0, a_1, \cdots, a_{k-1}$. First, using Gaussian elimination, which required $O(n^3)$ operations. Second, computing the inverse matrix of a square Vandermonde matrix

$$[a_0, a_1, \cdots, a_{k-1}]^T = \left(V_{ij}^{-1}\right)_{k\times k} \cdot [y_1, y_2, \cdots, y_k]^T \mod p,$$

where $a_{i-1} = \sum_{j=1}^{k} {V_{ij}}^{-1} y_j$ and evaluate:

$$V_{ij}^{-1} = \frac{(-1)^{i+1} \sum\limits_{\substack{1\le p_1<\cdots<p_{n-i}\le n \\ p_1,\ldots,p_{n-i}\neq j}} x_{p_1} x_{p_2} \cdots x_{p_{n-i}}}{\prod\limits_{\substack{1\le p\le n \\ p\neq j}} (x_p - x_j)}.$$

The implementation of reconstruction in FlashGhost as shown in Algorithm 2.

Algorithm 2 Message Reconstruction

Input : $shares$: at least k shares, p: the prime, n: the total number of shares
1: **for** $i = 0$ to $k-1$ **do**
2: $\quad a[i] \leftarrow LagrangeInterpolate(0, shares[i], p)$
3: **end for**
4: Concatenate $a[0], a[1], \cdots, a[k-1]$ into a single string M'
5: $M \leftarrow RemoveZeroPadding(M')$
6: **return** M

Security Evaluation: Secret sharing scheme is information-theoretically secure (and also perfectly secure) in that only sufficient shares can reconstruct the secret, while having less than the requisite number of shares provides no information about the secret [18].

D. Frequently Colliding Hash Table

Vanish's highlight is to leverage existing DHTs infrastructure. The three key properties: *Availability*, *Large-scale* and *Churn*, make DHTs extremely useful to a self-destructing data system. However, Vanish only make the decryption key disappear rather than destroy data stored in the server. So there is still a potential risk of being bleached.

We aim to design a hash table which has an inclination for frequently collisions to update values quickly in order to support the self-destructing message, called Frequently Colliding Hash Table (FCHT). Ideally, the hash table maps each keyword to a unique slot, and a collision occurs when two different inputs produce the same output. The traditional hash table tries to avoid collisions, while our FCHT intends to create collisions with a controllable rate. The former uses a dynamic resizing mechanism to reduce the possibility of collision, but the latter uses a fixed capability to allow collisions. Once a collision occurs, the new data will overwrite the old data. After writing the old data repeatedly with new data at the same position, we could consider the old data has been erased permanently, thereby realizing secure data destruction.

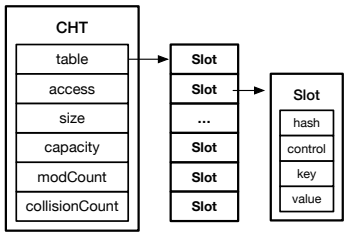


Fig. 3: The structure of FCHT in FlashGhost.

An example block structure of Frequently Colliding Hash Table is shown in Fig. 3. In this figure, besides the hash table, FCHT contains some specific properties. The property $access$ contains the access policy and the state of message (valid or invalid), which is updated dynamically, so that FlashGhost doesn't have to search all shares in the hash table to confirm whether a message is active. Others are the descriptive information of FCHT, $capacity$ is the number of slots in the table, $modCount$ is the total of message has been inserted (including the overwritten) and so on.

Algorithm 3 Put message

Input : M: message, K: keyword, C: access policy, H: FCHT
1: $index \leftarrow hash(K)$
2: $shares[] \leftarrow GenerateShares(M, C)$
3: **for** $i = 1$ to $n = shares.size()$ **do**
4: $\quad H[index + i] \leftarrow shares[i]$
5: **end for**
6: return

The core of the design of FCHT is two algorithms, Put and Get, i.e. $Insert$ and $Query$. These two algorithms, which present in Algorithm 3 and 4, could achieve the targets on erasing invalid message shares automatically, so as to support secure self-destruction of message.

Algorithm 4 Get message

Input : K: the keyword, H: FCHT
1: $index \leftarrow hash(K)$
2: $n, k \leftarrow$ Get parameters from first valid slot
3: **for** $i = 1$ to n **do**
4: $\quad$ **if** $S_i \leftarrow H[index + i]$ is not null **then**
5: $\quad\quad shares$.add(S_i)
6: $\quad$ **end if**
7: $\quad$ **when** $shares$.size() == k **break**;
8: **end for**
9: **if** S.size() < k **then**
10: $\quad$ **return** null
11: **else**
12: $\quad$ return $M \leftarrow$ *ReconstructMsg*($shares$)
13: **end if**

E. The Lifetime of Message

FlashGhost leverages secret sharing and hash collision to create two states for the degeneration of messages: valid and invalid, respectively. As shown in Fig. 4, for a certain message, its life cycle is divided into four stages.

Stage 1: ***Valid and Active***. Message is accessible to users at this stage, beginning from being created and end with being tagged invalid when $c \leqslant 0$ and $t > t_b$.

Stage 2: ***Invalid but Active***. The message has expired but less than $n-k$ shares have been overwritten by new shares in FCHT.

Stage 3: ***Degeneration***. At least $n-k+1$ shares have been overwritten one time, but it doesn't yet satisfy the secure removal of data.

Stage 4: ***Destroyed***. At least $n-k+1$ shares of the message have realized the secure data destruction, e.g., 7-pass, eliminating data remanence completely.

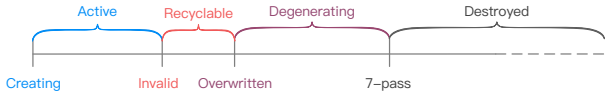


Fig. 4: Four stages of message's lifetime.

IV. Theoretical Analysis

In this section, we evaluate the performance of FlashGhost from theoretical analysis. Before providing the mathematical model, we first present some parameters that impact the performance of our system. These affecting factors are shown in Table I.

TABLE I: The influential parameters and evaluation indicators.

Influential parameters		**Evaluation indicators**	
the number of shares	n	survival probability	P_a
threshold	k	actual lifetime of message	d
insertion intensity	λ	execution time of an insertion	t_i
the capacity of FCHT	m	execution time of a query	t_q

Insertion intensity λ is a ratio of total insertion N_A and table capacity m, i.e., $\lambda = N_A/m$. By introducing λ, we reduce the parameter m to simplify our analyses. Survival probability is the percentage of a message still can be reconstructed after a certain period, which to evaluate the activeness of a message. Not like FullPP [19], which is able to control the life cycle of message that even before its predefined time. Considering the randomness of hash collision, our system can't exclude the possibility of reconstruction failure before message's invalidation. We define a message with $\Pr[Get(k) \neq null] > 99.9\%$ as an active message; a message with $\Pr < 1\%$ as an invalid message, and consider the period from $\Pr < 99.9\%$ to $\Pr \approx 0$ as the degeneration stage.

A. Mathematical Model

Suppose a given message M with a $(k,\ n)$-threshold scheme, which n shares have been stored in slots $S_1, S_2, \cdots, S_n$ in FCHT. The slot would not form a queue but a single block. Here we assume that a share of M is valid only if the queue length of the slot that stored it is zero, i.e. $L_S = 0$, where L_S denotes the queue length of a slot, meaning no collision over them at present. If a collision occurs in S, L_S plus one, so that an invalid share that has been overwritten can be understood as $L_S > 0$ of the slot stored.

With N_A insertions in total, the probability of a slot is overwritten i times can be expressed as:

$$\Pr[L_S = i] = \binom{N_A}{i}\left(\frac{1}{m}\right)^i\left(1-\frac{1}{m}\right)^{N_A-i}.$$

Let N_S denote the number of valid slots for M, so $\Pr[N_S = w]$ represents the probability that M has w valid slots in FCHT. When N_A gets very large, the above binomial distribution can be expressed as a Poisson distribution:

$$\begin{aligned}\Pr[N_S = w] &= \binom{n}{w}\Pr[L_S = 0]^w \Pr[L_S \geqslant 1]^{n-w} \\ &= \binom{n}{w}\left(1-\frac{1}{m}\right)^{N_A w}\left[1-\left(1-\frac{1}{m}\right)^{N_A}\right]^{n-w} \\ &\approx \binom{n}{w} e^{-\frac{N_A w}{m}}[1-e^{-\frac{N_A}{m}}]^{n-w}.\end{aligned}$$

Concerning any k shares can reconstruct M, we can use $\Pr[N_S \geqslant k]$ to represent the survival probability of message. Equation 3 shows the probability that M can be reconstructed at a certain time t.

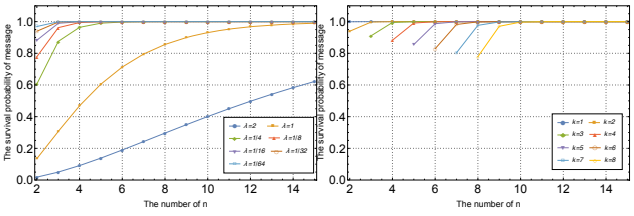
$$\Pr[N_S \geqslant k] = \sum_{w=k}^{n}\binom{n}{w} e^{-\lambda t w}[1-e^{-\lambda t}]^{n-w} \tag{3}$$

Now we get our mathematical model, then we aim to control the lifetime and degeneration process of message by adjusting the value of parameters in the model.

B. Case studies

An observation derived from Equation 3 lead us to know the availability of message is affected by parameters, such as the number of shares n, threshold k and insertion intensity λ. For example, the lifetime of a message is x times longer when $\bar{n}$ and k remain unchanged and λ changes into λ/x. Our goal is to assess how the survival probability is affected by these factors. To investigate the impacts of different parameters on the activeness and degeneration of messages, we use the above mathematical model to conduct several experiments by controlling variable. By plotting experimental results into graphs, we can clearly see the life cycle and degeneration process of messages.

1) Over-collision: In an FCHT with fixed capacity m, when there are q insertions within a time unit (e.g. second), the collision rate of each slot is q/m. Hence, it's possible that a message becomes invalid just after being inserted if the number of shares is less than q. Therefore, we here discuss the probability that successful first read of a message within its validity period. It's easy to know that the higher insertion intensity, the lower success rate to reconstruct M. If a message expires just after being inserted or before an expected life, we call this phenomenon an over-collision.



(a) Survival probability with insertion intensity λ.

(b) Survival probability with threshold k.

Fig. 5: The survival probability of message over different number of shares when $t = 1$s.

In Fig. 5 we show the survival probability of message over the different number of n when $t = 1$s. The first line graph compares the different insertion intensity ($\lambda = 2, 1, 1/4, 1/8, 1/16, 1/32, 1/64$) from $n = 2$ to $n = 14$ when $k = 2$ in terms of the survival probability of message. It can be drawn from the graph that the survival probability P_a rises as the number of n increases and as the insertion intensity decreases. It is noticeable that P_a increases dramatically with decreasing insertion intensity from 2 to $1/4$. Accordingly, the second graph compares different threshold ($k = 1, 2, 3, 4, 5, 6, 7, 8$) from $n = 2$ to $n = 14$ with $\lambda = 1/32$. We can see P_a decreases as threshold increases.

2) The Number of Shares n: It's not hard to know that the larger number of shares supports longer timeout, because the system can tolerate more share loss. In order to evaluate the impacts of the number of shares on the lifetime of the message, we choose n to $4, 8, 16, 32, 64, 128, 256$, respectively with $k = 2$ and $\lambda = 1/32$, to observe the survival probability over a period of 300 seconds. From Fig. 6, we see that increasing n improves availability. The lifetime of a message gets longer when we have more shares. Furthermore, for $n = 2^x$, the increment of x extends 20 seconds more to the lifetime of message. Besides, the number of shares almost has no effect on the degeneration process. As shown in the Fig. 6, from $Pr < 1$ to $Pr \approx 0$, the total time of degeneration stays at around 120 seconds except for $n = 4$ with 100 seconds. It indicates that the time of message degeneration does not be extended or shortened as n increases.

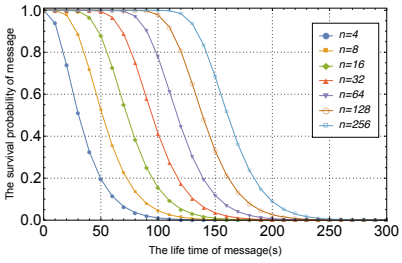


Fig. 6: The survival probability with increasing n.

3) Threshold k: Similarly, the smaller threshold also supports a longer lifetime of the message, because it makes reconstruction of it easier. The larger k is, the more shares are required to be fetched, and the lower probability of successful reconstruction. In order to evaluate the impacts of threshold on the lifetime of a message, we set k to $1, 2, 3, 4, 5, 6, 7$ and 8, respectively, with $n = 16$ and $\lambda = 1/32$, to observe the survival probability over a period of 300 seconds.

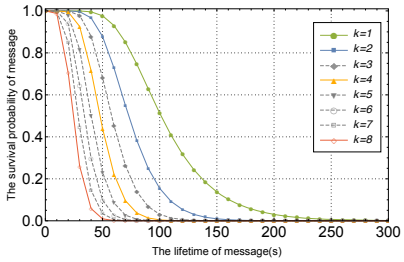


Fig. 7: The survival probability with increasing k.

In Fig. 7, we show that k has an impact on both the lifetime and degeneration of message, especially on the degeneration process. We see the degeneration period decreases from 123 seconds to 73 seconds when we increase k from 2 to 4. In this figure, we use coloured solid lines to denote an exponential growth of k with the base 2, while the grey dashed lines to indicate k is incremented by 1. We can see that the degeneration process of the message changes uniformly with the exponential growth of k.

4) insertion intensity λ: For a fixed-capacity FCHT, the insertion intensity of the message determines the frequency of hash collision. Higher insertion intensity brings higher collision frequency, which decreases the availability. We conduct nine groups of experiments with different insertion intensity, $\lambda = 1, 1/8, 1/16, 1/24, 1/32, 1/48, 1/56$, and $1/64$, respectively, with $n = 16$ and $k = 2$.

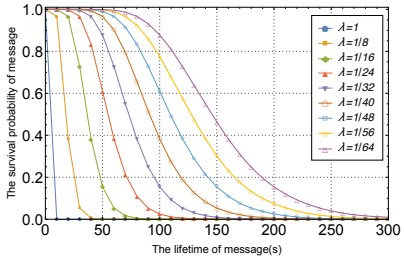


Fig. 8: The survival probability with decreasing λ.

The results are plotted in Fig. 8. We can see that the effect of insertion intensity on the message is similar to that of k, which has an impact both on validity period and degeneration process of a message, especially the degeneration process. In the experiments, we use the step size of 10 seconds to plot each curve. Taking a look at the curve with $\lambda = 1/8$, we can see that the main degeneration process lasts around 20 seconds, from 10 s to 30 s (see the second orange line). Likewise, the main degeneration processes of $\lambda = 1/16, 1/24, 1/32 \cdots, 1/64$ are $40, 60, 80, \cdots, 160$ seconds, respectively, which present a linear growth. Besides, we also can see the lifetime of a message increases evenly as the insertion intensity decreases, which the lifetime of a message with $\lambda = 1/8, 1/16, 1/24, 1/32, 1/40$ are around $6, 12, 19, 25, 31$ seconds, respectively. It demonstrates that the insertion intensity is important to control both degeneration and activeness period of message.

5) FCHT Capacity m: In above analyses, we introduce parameter λ (insertion intensity) to reduce two parameters (N_A

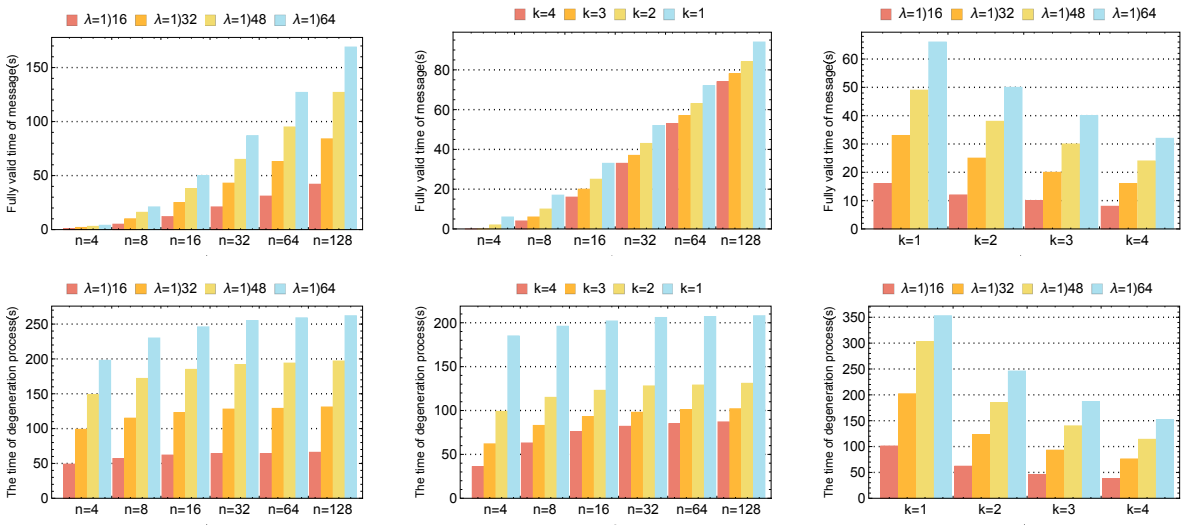


Fig. 9: Six cases of message lifetime and degeneration with various two-parameter combination; the default value of parameters are $k = 2$ (a, d), $\lambda = 1/32$ (b, e) and $n = 16$ (c, f).

and m) into a single one. However, in the actual application, table capacity plays an important role in message's survival probability. We say that FCHT in our system is fixed-capacity, which doesn't mean the value of capacity is immutable. Not surprisingly, with the same amount of insertions, the message in a system with larger capacity lives longer than a smaller capacity. That is to say, the value of m depends on the data throughput. Higher throughput requires larger capacity, and vice versa. To a certain scenario, the ideal value can be computed by our mathematical model.

6) General Discussion: The foregoing analyses discuss the impact of a single parameter on message's lifetime and degeneration. We put it all together to see a combined impact on message. The six bar graphs in Fig. 9 give information and compare the influence of any two-parameter combinations on lifetime and degeneration of message.

V. Performance Evaluation

In this section, we evaluate the performance of FlashGhost from experimental verification. Unlike theoretical analysis, it is inevitable that there are some disturbing factors to degrade the practical performance. On the hardware side, CPU speed, memory, time accuracy, etc, should be taken into consideration, and on the software side, the system is subjected to the code logic, algorithm, etc. Hence, we conduct extensive experiments to compare practical system performance with above mathematical model.

A. Experimental Setup

To confirm our analysis, we have developed a simulation application written in Java. The configuration of this application is summarized in Table II.

TABLE II: Test Environment Configuration.

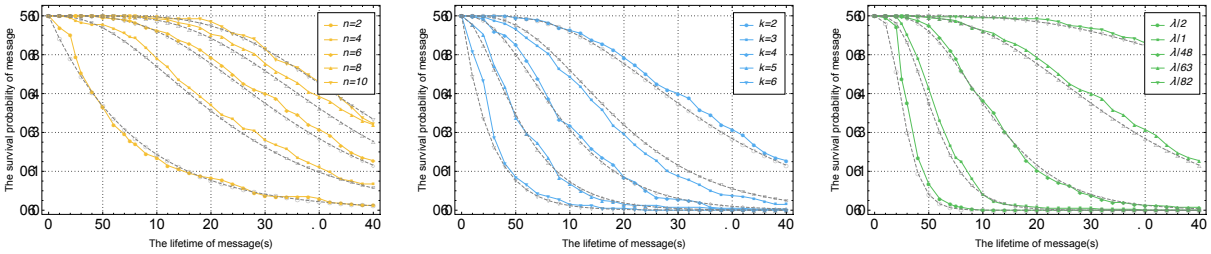
Hardware and System	
CPU	2.6 GHz Intel Core i5
Memory	8 GB
OS	macOS Sierra
Development	
Program Language	Java (JDK1.8)

B. Experimental Verification

Similar to theoretical analyses in the previous section, we conduct three groups of experiments, which each group has tested over 100,000 messages, to discuss the effects of three parameters (n, k and λ) on the survival probability of message, respectively. The experimental results are plotted in Fig. 10. In this figure, the coloured solid lines denote experimental results and the grey dashed lines indicate theoretical results. Comparing two situations on each figure, we can see the practical results fit in with the theoretical results nicely.

C. Time Load

The main time load of our system depends on the encryption and decryption of the message. There are two key influencing factors, one is message length len and the other is threshold k. The general Shamir's secret sharing scheme is to place the whole secret in the first coefficient of the polynomial. Although this method can be quickly reconstructed, once the secret becomes long, the encryption takes a long time. The reason is that the longer secret, the larger prime must be picked. Therefore, prime generation and testing become the biggest time overhead. Our scheme takes a different approach, which allows the message length to be longer than the prime length by splitting M into shorter pieces. We conducted six groups of experiments with different message lengths to test the total overhead for both encryption and decryption in both FlashGhost and traditional Shamir's secret sharing scheme. In the experiment, we tested 64, 120, 240, 800, 1600, and 3200-bit messages, respectively. For each group, we test 200 messages, which contains different threshold schemes (for k from 1 to 10) for getting comprehensive results. The experimental results are shown in Table III, where $En.$ and $De.$



(a) $k = 2,\ \lambda = 1/32$ (b) $n = 6,\ \lambda = 1/32$ (c) $n = 6,\ k = 2$

Fig. 10: The programming experiment results of FlashGhost: (a) shares n, (b) threshold k, (c) insertion intensity λ.

denote the time of encryption and decryption (reconstruction) in milliseconds, respectively.

TABLE III: Time load of different secret sharing schemes.

length	64	120	240	800	1600	3200
Shamir's secret sharing scheme with secret as the first coefficient						
En. (ms)	33.56	49.68	84.44	153.58	1070.37	2485.22
De. (ms)	16.18	29.76	30.62	27.05	23.88	20.77
FlashGhost's multi-message secret sharing scheme						
En (ms)	3.25	1.39	4.38	12.16	61.52	106.26
De (ms)	1.81	0.97	0.92	1.14	0.74	2.14

The results show that our scheme has better performance than the traditional schemes. Considering we only conducted experiments with thresholds from 1 to 10, when the message is very long, the encryption takes more time as the length increases. For a long message, it is advisable to break it into more pieces in order to reduce the time overhead. Also, we can see that the overhead of decryption has no correlation with the length of the message.

VI. Conclusion

In this paper, we propose an automatic data sanitization system called FlashGhost, which aims to make data self-destruct after being useless to users, without any actions by users and any centralized or trusted system. We present the design of FlashGhost where a novel aspect of our approach is the integration of threshold cryptography with a frequent colliding hash table. We prove that our solution is secure once the message reaches invalid state so as to anyone, including the sender and server, cannot retroactively obtain the expired messages, through legal or other means. Our theoretical analyses show the feasibility of our approach and the experiments also demonstrate its efficiency and practicality.

Acknowledgment

This work is supported by the National Key Technologies R&D Programs of China (Grant No. 2018YFB1402702), NSFC-Genertec Joint Fund For Basic Research (Grant No. U1636104) and Joint Research Fund for Overseas Chinese Scholars and Scholars in Hong Kong and Macao (Grant No. 61628201).

References

[1] Miniwatts Marketing Group, "World internet users and 2017 population stats," 2017. [Online]. Available: http://www.internetworldstats.com/stats.htm

[2] ——, "Internet growth statistics," 2017. [Online]. Available: https://www.internetworldstats.com/emarketing.htm

[3] American Academy of Matrimonial Lawyers, Wednesday, "Huge increase of texts and app evidence in divorces say nation's top lawyers," 2015. [Online]. Available: http://aaml.org/about-the-academy/press/press-releases/divorce/huge-increase-texts-and-app-evidence-divorces-say-nat

[4] Rosenberg Matthew; Confessore Nicholas; Cadwalladr Carole, "How trump consultants exploited the facebook data of millions," 2018. [Online]. Available: https://www.nytimes.com/2018/03/17/us/politics/cambridge-analytica-trump-campaign.html

[5] R. Perlman, "The ephemerizer: Making data disappear," 2005.

[6] Y. Tang, P. P. Lee, J. C. Lui, and R. Perlman, "Fade: Secure overlay cloud storage with file assured deletion," in *International Conference on Security and Privacy in Communication Systems*. Springer, 2010, pp. 380–397.

[7] S. Nepal, C. Friedrich, C. Wise, R. O. Sinnott, J. Jang-Jaccard, and S. Chen, "Key management service: Enabling secure sharing and deleting of documents on public clouds," *Services Transactions on Cloud Computing (STCC)*, vol. 4, no. 2, p. 2016, 2016.

[8] R. Geambasu, T. Kohno, A. A. Levy, and H. M. Levy, "Vanish: Increasing data privacy with self-destructing data." in *USENIX Security Symposium*, vol. 316, 2009.

[9] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[10] L. Zeng, Z. Shi, S. Xu, and D. Feng, "Safevanish: An improved data self-destruction for protecting data privacy," in *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*. IEEE, 2010, pp. 521–528.

[11] G. Wang, F. Yue, and Q. Liu, "A secure self-destructing scheme for electronic data," *Journal of Computer and System Sciences*, vol. 79, no. 2, pp. 279–290, 2013.

[12] 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 *NDSS*, 2010.

[13] N. RamaKalpana and R. Santhosh, "Sedas self-destruction data system for distributed object based active storage framework," *International Journal of Software and Web Sciences, 7 (1), December 2013*, pp. 94–100, 2014.

[14] US Department of Commerce, NIST, "Nist special publication 800-88, revision 1: Guidelines for media sanitization," *Itlb*.

[15] J. Reardon, D. Basin, and S. Capkun, "Sok: Secure data deletion," in *Security and Privacy (SP), 2013 IEEE Symposium on*. IEEE, 2013, pp. 301–315.

[16] J. Xiong, X. Liu, Z. Yao, J. Ma, Q. Li, K. Geng, and P. S. Chen, "A secure data self-destructing scheme in cloud computing," *IEEE Transactions on Cloud Computing*, vol. 2, no. 4, pp. 448–458, 2014.

[17] Y. Zhu, L. Yang, and D. Ma, "Secure snaps: a new forward secrecy cryptosystem for self-destructing messages in mobile services," in *Mobile Services (MS), 2015 IEEE International Conference on*. IEEE, 2015, pp. 142–149.

[18] T. P. Pedersen *et al.*, "Non-interactive and information-theoretic secure verifiable secret sharing." in *Crypto*, vol. 91, no. 7. Springer, 1991, pp. 129–140.

[19] J. Xiong, F. Li, J. Ma, X. Liu, Z. Yao, and P. S. Chen, "A full lifecycle privacy protection scheme for sensitive data in cloud computing," *Peer-to-peer Networking and Applications*, vol. 8, no. 6, pp. 1025–1037, 2015.