# Timed-release of Self-emerging Data using Distributed Hash Tables

Chao Li
School of Information Sciences
University of Pittsburgh
Pittsburgh, USA
Email: chl205@pitt.edu

Balaji Palanisamy
School of Information Sciences
University of Pittsburgh
Pittsburgh, USA
Email: bpalan@pitt.edu

***Abstract*—Releasing private data to the future is a challenging problem. Making private data accessible at a future point in time requires mechanisms to keep data secure and undiscovered so that protected data is not available prior to the legitimate release time and the data appears automatically at the expected release time. In this paper, we develop new mechanisms to support self-emerging data storage that securely hide keys of encrypted data in a Distributed Hash Table (DHT) network that makes the encryption keys automatically appear at the predetermined release time so that the protected encrypted private data can be decrypted at the release time. We show that a straight-forward approach of privately storing keys in a DHT is prone to a number of attacks that could either make the hidden data appear before the prescribed release time (*release-ahead attack*) or destroy the hidden data altogether (*drop attack*). We develop a suite of self-emerging key routing mechanisms for securely storing and routing encryption keys in the DHT. We show that the proposed scheme is resilient to both *release-ahead attack* and *drop attack* as well as to attacks that arise due to traditional churn issues in DHT networks. Our experimental evaluation demonstrates the performance of the proposed schemes in terms of attack resilience and churn resilience.**

## I. Introduction

In the age of Big Data, releasing private data to the future is a challenging problem. Making private data accessible at a future point in time requires mechanisms to keep data secure and undiscovered so that protected data is not available prior to the legitimate release time and the data appears automatically at the expected release time. Such self emerging data are not allowed to be accessed immediately after they are produced. Examples of such data include private data of individuals with privacy requirements that degrade over time [11]. For example, personal data of individuals (e.g., medical diagnostics information, web browsing patterns, location trajectory patterns) collected during their lifetime may be highly sensitive during the childhood and youth life of an individual, however, the same data may become less sensitive as the individual ages and after the end of the individual's life. Other examples of self emerging data include incorporating a secure voting mechanism where the encrypted votes of individuals may be collected during the polling process but the data may be allowed to be accessed (decrypted) only after the end of the polling process (*release time*). Similarly, an online examination scheduled to be administered at a given time (*release time*) may not be accessed before the prescribed start time.

Supporting self emergence of data involves encrypting the data and ensuring that the encryption key is destroyed and remains unavailable until the release time. The encryption key automatically appears at the release time and makes the data *self-emerge* at the release time. A straight-forward approach to implementing self-emergence of data would be to store the encryption key on a trusted third party server which protects the encryption key until the release time and makes it available precisely at the release time. However, such a straight-forward approach suffers from a single point of trust. An adversary in such an approach can locate the key and focus on breaching the security of the trusted server to release the data prior to the release time which can violate the intended privacy provided by self-emerging data.

In this paper, we develop a highly distributed solution for supporting self-emerging data using Distributed Hash Table (DHT) networks [20] that prevent the adversary from obtaining the shares of the encryption key dispersed in the DHT before the legitimate release time. The proposed mechanisms route the encryption key on the DHT in a deterministically pseudo-random manner making it automatically appear at the release time while making it harder for the adversary to access it prior to the release time. Compared to a centralized key storage scheme, the proposed approach using large-scale DHT networks significantly increase the attack resilience offered by the scheme.

The proposed approach for self emergence of data develops a novel integration of cryptographic techniques with DHTs to enable self emergence of protected data at the release time. We note that the proposed approach requires a strong defensive performance towards powerful adversaries controlling a fraction of the nodes in the DHT to forcibly extract the key before the timeout. Such adversaries can sabotage our intended goal in multiple ways leading to attacks that could either make the hidden data appear before the prescribed release time (*release-ahead attack*) or destroy the hidden data altogether (*drop attack*). In this paper, we present a suite of self-emerging key routing mechanisms on DHTs for securely storing and routing encryption keys of self-emerging data. The proposed techniques use a novel and effective combination of onion

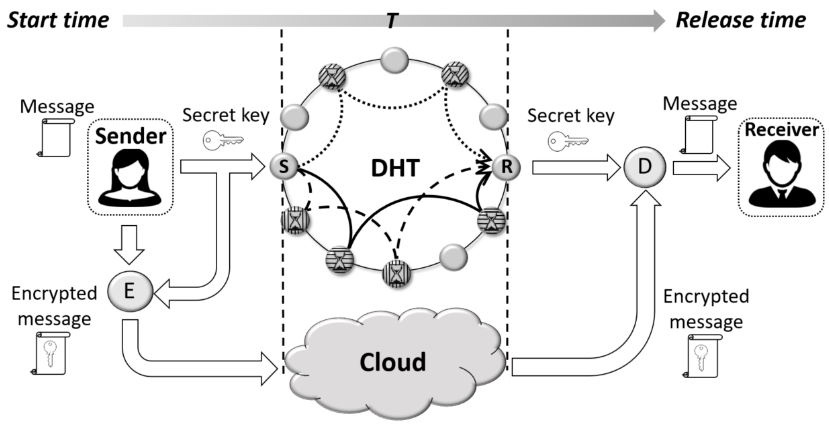


Fig. 1: Private data timed release system

routing [14], data replication and Shamir secret sharing [17] mechanisms to guarantee a higher degree of attack resilience and performance.

In the rest of the paper, we introduce the self-emerging data timed-release system and analyze its challenges in Section II. In Section III, we present the proposed approach to handle the challenges and discuss the routing path pattern construction algorithms. In section IV, we evaluate our approach in terms of attack resilience and churn resilience. Finally, we present the related work in section V and conclude in section VI.

## II. System model and challenges

In this section, we present the proposed self-emerging data release system, followed by the challenges including the adversary models and the churn issues in DHTs.

### A. Self-emerging data timed-release system

The proposed self-emerging data release system consists of four major entities, namely the data sender, the data receiver, the DHT network and the cloud, shown in Figure 1.

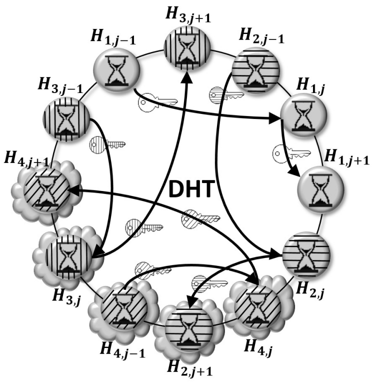
- **Data sender**: The data sender is the entity that wants to send data to the data receiver. Specifically, she wants to send a message out at start time $t_s$ and allow it to be accessible by the data receiver only after release time $t_r$, where $t_s < t_r$. In our system, the data sender, Alice, encrypts her message with a secret key and then sends the secret key and encrypted message to the DHT and the cloud respectively at start time. After $t_s$, there is no any further involvement required from Alice.
- **Data receiver**: The data receiver is the entity that wants to get access to the message sent by the data sender. In our system, the data receiver, Bob, can get the encrypted message from the cloud at any time after $t_s$. However, he should be allowed to get the secret key from the DHT only after the data release time, $t_r$ so that the plain text message is only available to him after $t_r$. Bob may start to extract the secret key from the DHT before $t_r$, which makes him an adversary in that case.
- **DHT network**: A DHT network is required to hold and hide the secret key during the time period $t_r - t_s$, defined as the emerging time period $T$. To hold the secret key, during $T$, the DHT network should keep the existence of the secret key and prevent the loss of it. To hide the secret key, the DHT network should prevent the secret key from being found and extracted by any entities, including Bob, before $t_s$. To realize these, we design a suite of mechanisms in Section III, which carefully determine routing paths for the secret keys to be transmitted from the source to destination with high attack resilience.
- **Cloud**: A cloud is required to store the encrypted data during $T$. The encrypted data is always accessible by the authenticated data receivers.

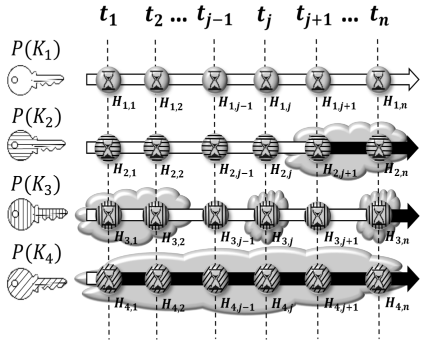
### B. Adversary models

An entity is considered to be an adversary when it tries to block the ‘hold and hide’ responsibility of the DHT protocol in the system. That is, it tries to ‘steal or drop’ the secret key stored in the DHT during the emerging time $T = t_r - t_s$. From this perspective, even Bob can be an adversary. To execute an attack, a fraction of nodes (for example $p$ percentage) in the DHT network are required to be malicious, which can be realized through Sybil attack [5] or Eclipse attack [18] performed either by a single adversary or through the collusion among a group of adversaries. For example, Sybil attack typically works as an amplifier to boost the destructive power of the system-targeted attack models. The adversary may create a large number of pseudonymous identities and using them to gain a disproportionately large influence.

In this paper, we present two potential attack models, namely the release-ahead attack aiming to steal the secret key from the DHT before the release time $t_r$ and the drop attack aiming to destroy the secret key so that the data cannot be restored after $t_r$.

*1) Release-ahead attack:* The release-ahead attack aims to furtively extract the secret key from the DHT before the release time, $t_r$ and uses it to decrypt the encrypted data stored in the cloud, thus compromising the confidentiality of the private data. For example, in the scenario of an online examination released using self emerging data, if one of the participants attacks the system by extracting the secret key from the DHT network before the examination start time, she can download


(a) Secret key paths in DHT


(b) Release-ahead attack

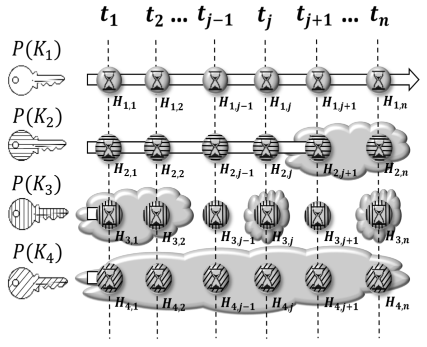

(c) Drop attack

Fig. 2: Attack models

the encrypted test questions from the cloud and decrypt it with the key to leak the examination questions in advance.

Figure 2(a) shows a simplified example of the behavior of four secret keys in the DHT. In brief, during the emerging time T from $t_s = t_1$ to $t_r = t_n$, each secret key sequentially moves along a pre-determined onion routing [14] path formed by a set of DHT nodes (called holders) and stop over at each holder for a period of time. For this example, we denote the $j^{th}$ holder on the path of $i^{th}$ secret key as $H_{i,j}$ and the path of $i^{th}$ secret key as $P(K_i)$. Before launching the attacks, we assume a fraction of nodes in DHT have been controlled by the adversary through the traditional Sybil attack, including holders $H_{2,j+1}$, $H_{3,j}$, $H_{4,j-1}$, $H_{4,j}$ and $H_{4,j+1}$. In Figure 2(b), we analyze the release-ahead attack result on the four secret keys. The first secret key $K_1$ can be securely received by the receiver because its path has no malicious holder. On the path of $K_2$, the last two holders are malicious, so $K_2$ can be released at time $t_{j+1}$. The path of $K_3$ has malicious holders in the head, middle and tail, but $K_3$ cannot be released before $t_n$ because the adversary does not have the key stored on holder $H_{3,j+1}$ to decrypt that layer of the onion. Finally, the $K_4$ can be released at the beginning $t_1$ because all the nodes on the path are malicious. To sum up, the onion structure forces the adversary to control a set of successive holders starting from the last one on the path. Any break in the continuity stops the release-ahead attack from releasing the secret key prior to release time.

*2) Drop attack:* The objective of the drop attack is to make the secret key unavailable at the start time, $t_r$, thus compromising the availability of the self-emerging data when it is required to be released. Typically, a successful drop attack means the loss of the private self-emerging data as the secret key can not be restored to decrypt the encrypted data. In the online examination example, a successful drop attack means that the examination questions are destroyed once for all and are never readable.

Figure 2(c) shows the drop attack results on the four secret keys for the example shown in Figure 2(a). To drop the secret key, whenever a holder controlled by the adversary receives it, the holder deliberately refuses to pass it to the next holder along the path. Therefore, $K_1$ is safe but all the other three secret keys are dropped because there is at least one malicious holder on their paths. As can be seen, the current path structure is very vulnerable to drop attack as it is very hard to have a non-malicious path under strong Sybil attack. In Section III, we will propose a suite of strategies to enhance the drop attack resilience.

### C. Churn impact

So far in the discussion, we have simply assumed that all the nodes in the DHT are stable and the only threat in the DHT system are the malicious nodes. However, in practice, this is not entirely true due to the fact that DHT networks are prone to the churn issue [21]. The churn in the DHT has short-term and long-term impacts to the self-emerging data stored in the system. Churn can be briefly summarized as node unavailability and node death respectively. A node in a DHT may leave the network transiently due to network failure, instability, or owner personal issues and rejoin the network after the short departure. Because of node unavailability, at release time $t_r$, the secret key may not be entirely available to be released on time as the transmission might be blocked by some unavailable nodes on path. In a DHT, a node may also leave the network forever due to hardware problems or loss of interest. This leads to the 'dead' time of the node in its lifetime, as the node ID in the DHT is no longer retained and the stored data is lost. Because of the death of the nodes, the secret key stored on it may also be lost. Even if the key is replicated to another node through the replication mechanism, we note that the new node also has probability $p$ (node malicious rate) to be malicious, thus increasing the chance for the adversary to steal this key.

The above analysis of the challenges and attacks motivates us to the design of the proposed self-emerging key routing scheme in Section III that achieves a high degree of attack resilience to both the adversarial attack models and the churn issue in DHTs.

## III. Self-emerging key routing in DHT

Our self-emerging key routing schemes consist of three components namely a routing path construction scheme, a

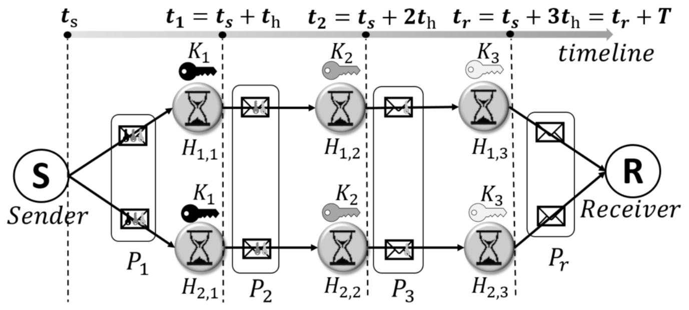


Fig. 3: Node-disjoint multipath routing

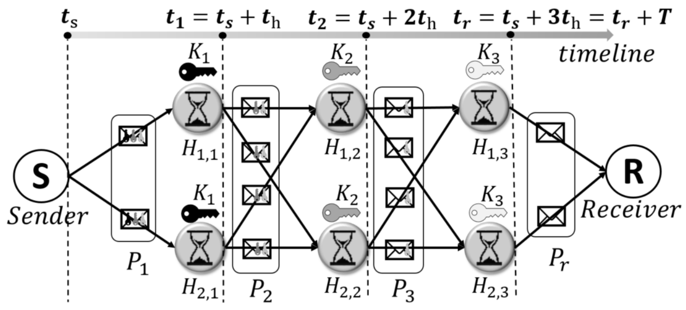


Fig. 4: Node-joint multipath routing

package generation scheme and package transmission protocol. Specifically, the secret key owner initially joins the DHT network and pseudo-randomly selects nodes in the DHT to form the routing paths using the routing path construction scheme. Then, the owner locally encapsulates the secret key in transmission packages. Finally, the packages are routed along the paths to their destinations, they stop at each passed nodes (named holders) for a time period $t_h$ (named holding period) so that the secret key can be finally released at release time, $t_r$ by the terminal nodes on the paths.

In this section, we propose and analyze four self-emerging key routing schemes. We start from the centralized scheme, which stores the secret key on a single node for the entire emerging time $T$. Then, by applying onion routing and replication mechanisms, we design the node-disjoint multipath routing and node-joint multipath routing schemes to improve the attack resilience for shorter self-emerging time $T$. Finally, to suppress the churn impact for long self-emerging time $T$, we replace the replication mechanism with a key share delivery scheme based on Shamir secret share [17] to enhance churn resilience. To compare the schemes in terms of attack resilience, we measure the release-ahead attack resilience, $R_r$ as the probability that an adversary fail to restore the secret key at the start time $t_s$ by collecting necessary data from the malicious nodes, and drop attack resilience, $R_d$ as the probability that an adversary fail to prevent the secret key to be released at the release time $t_r$ by dropping data through the malicious node.

## A. Centralized scheme

The centralized scheme uses a single DHT node to store the secret key for the entire emerging period $T$. This simple scheme has the lowest attack resilience because the adversary can obtain the secret key at start time $t_s$ and choose to either release it or drop it if this node is malicious, which has the probability $p$ (node malicious rate). Therefore, both the release-ahead attack resilience, $R_r$ and drop attack resilience $R_d$ is $1 - p$ in this scheme.

## B. Node-disjoint multipath routing scheme

The node-disjoint multipath routing scheme applies onion routing mechanism [14] to improve release-ahead attack resilience, $R_r$ and employs a replication scheme to improve drop attack resilience, $R_d$. In the example shown in Figure 3, there are 2 replicated node-disjoint onion routing paths, $Sender \rightarrow H_{1,1} \rightarrow H_{1,2} \rightarrow H_{1,3} \rightarrow Receiver$ and $Sender \rightarrow H_{2,1} \rightarrow H_{2,2} \rightarrow H_{2,3} \rightarrow Receiver$ with 3 holders on each of them, where a holder $H_{i,j}$ denotes the $j^{th}$ holder on the $i^{th}$ path. At start time $t_s$, the sender locally generates onion $P_1$ with three keys $K_1$, $K_1$ and $K_3$, assigns the keys to the six holders as shown and sends $P_1$ to two holders $H_{1,1}$ and $H_{2,1}$, namely the first holders on the two paths. Once the two holders receive $P_1$, they decrypt one layer of the onion $P_1$ with key $K_1$ to get the IDs of the second holders on the paths and the remaining onion $P_2$, and hold $P_2$ for a holding period $t_h = \frac{T}{3}$. Subsequently, at time $t_1 = t_s + t_h$, $H_{1,1}$ and $H_{2,1}$ send the remaining onion $P_2$ to the second holders, namely $H_{1,2}$ and $H_{2,2}$. The second holders decrypt the next layer of $P_2$ with key $K_2$ to get the IDs of the third holders on the paths and the remaining onion $P_3$. After they hold $P_3$ for another holding period $t_h$, at time $t_2 = t_s + 2t_h$, $P_3$ is sent to third holders $H_{1,3}$ and $H_{2,3}$. The third holders decrypt $P_3$ with $K_3$ and get the secret key. They will hold the secret keys for a last holding period $t_h$ to make the entire emerging period to be the expected value, $T = 3 * t_h$ and then release the secret key at the release time $t_r = t_3 = t_s + 3t_h$ to the receiver. In order to obtain the key at start time $t_s$, the adversary need to control at least one holder between every two holders with the same key to collect $K_1$, $K_2$ and $K_3$ to decrypt all the three layers of the onion $P_1$, which gives the release-ahead attack resilience:

$$R_r = 1 - (1 - (1 - p)^k)^l \tag{1}$$

where $k$ denotes the number of replicated node-disjoint onion paths and $l$ denotes the number of holders on each path. To prevent the secret key to be released at the release time $t_r$, the adversary has to control at least one holder among the three holders of each path to cut both the two paths, which gives the drop attack resilience:

$$R_d = 1 - (1 - (1 - p)^l)^k \tag{2}$$

The sender can apply equations 1 and 2 to calculate $k$ and $l$ and also $t_h = \frac{T}{l}$ for her expected attack resilience $R_r$ and $R_d$ with a known node malicious rate $p$, namely the ratio of malicious nodes in DHT. The sender can then pseudo-randomly select IDs of the $k * l$ holders to generate the onion $P_1$, assign the keys to the holders and send $P_1$ out to the first holders. Each holder will decrypt one layer of the onion with its key, hold the remaining onion for the holding period $t_h$ and forward it to the next holder at the end of $t_h$. Finally, the secret key can be sent to the receiver at release time $t_r$ by the terminal holders on the paths.

## C. Node-joint multipath routing scheme

The node-joint multipath routing scheme is similar to the node-disjoint multipath routing scheme but allows the onion routing paths to have intersecting holders. To maximize the drop attack resilience $R_d$, the node-joint multipath routing scheme connects every $j^{th}$ holder on the paths to every $(j+1)^{th}$ holder on the paths. The node-joint multipath routing shown in Figure 4 provides the extension of the example shown in Figure 3. We find that the number of onion routing paths has been increased to 8, which has been maximized without changing the replication number of $K_1$, $K_2$ and $K_3$. Thus, without impacting the release-ahead attack resilience $R_r$, the scheme reduces the drop attack resilience $R_d$ as the adversary now has to control all the holders holding the onion at the same time, namely the group $(H_{1,1}, H_{2,1})$, or $(H_{1,2}, H_{2,2})$, or $(H_{1,3}, H_{2,3})$ to drop the secret key. For instance, if the holders $(H_{1,1}, H_{2,2}, H_{1,3})$ are malicious, the adversary can drop the secret key in a node-disjoint multipath routing scheme but she can not do so in the node-joint multipath routing scheme as the path $Sender \rightarrow H_{2,1} \rightarrow H_{1,2} \rightarrow H_{2,3} \rightarrow Receiver$ is still alive. This feature makes:

$$R_d = (1 - p^k)^l \tag{3}$$

which can be combined with equations 1 to calculate the replication factor $k$, path length $l$ and the holding period $t_h$. We can also derive:

*Lemma 1: The node-joint multipath routing scheme can guarantee $R_r + R_d > 1$ when $p < 0.5$.*

*Proof:* $p < 0.5 \rightarrow p < 1-p \rightarrow 1-p^k > 1-(1-p)^k \rightarrow (1-p^k)^l > (1-(1-p)^k)^l \rightarrow R_d > 1-R_r \rightarrow R_r + R_d > 1$. Therefore, if we set $R_r = R_d$ to expect the same release-ahead resilience and drop attack resilience, we can get $R_r = R_d > 0.5$ for $p < 0.5$. In other words, the node-joint multipath routing scheme can make both $R_r$ and $R_d$ higher than 0.5 when $p < 0.5$. It also indicates the tradeoff between $R_r$ and $R_d$ and the relationship between the tradeoff and $p$, which helps to design a highly attack-resilient system.

## D. Key share routing scheme

The node disjoint and node joint multipath routing schemes require the sender to send the keys $K_1$, $K_2$ and $K_3$ to the holders at start time $t_s$ and hence the terminal holders on the paths, namely $H_{1,3}$ and $H_{2,3}$, have to store the key $K_3$ for nearly the entire emerging period $T$. This approach is efficient when $T$ is small, indicating that the keys for onion decryption do not need to be stored for a long time. However, when $T$ is large, the probability that the adversary can get the stored keys, such as $K_3$ in the example, is significantly increased due to churn. For instance, if holder $H_{1,3}$ is dead, a new node will take the place of $H_{1,3}$ and receive the replication of $K_3$ from the $H_{2,3}$ that is alive. This process is equivalent to providing

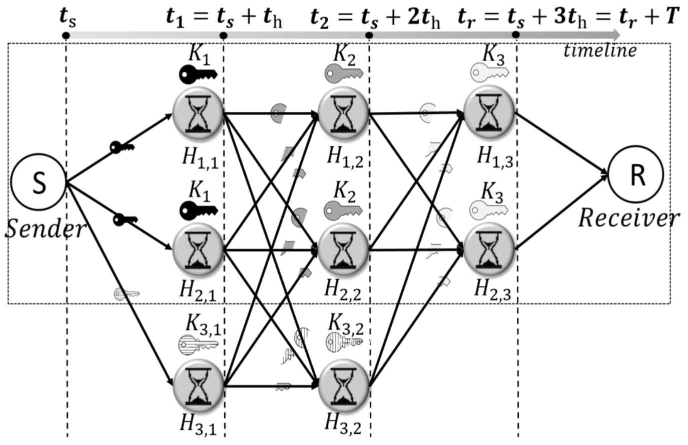


Fig. 5: Key share routing

the adversary another opportunity to obtain the stored key ($K_3$ in the example) as the new node has probability $p$ to be malicious. As a result, the number of nodes knowing the stored key, $K_3$ is increased from 2 to 3 and the chance for the adversary to get stored key $K_3$ is increased from $1-(1-p)^2$ to $1-(1-p)^3$, thus making release-ahead attack easier to be successful.

To increase the churn resilience, instead of pre-assigning the keys to the holders at start time $t_s$, the proposed key share routing scheme builds another layer of paths to route the keys for onion decryption to their target holders at the same time when the onions arrives. For example, in Figure 4, we want the holder $H_{1,3}$ to receive both the onion $P_3$ and key $K_3$ at the time $t_2$ so that $H_{1,3}$ does not need to store $K_3$ from start time $t_s$ to $t_2$. To make that possible, the proposed key share routing scheme applies the Shamir secret share [17] mechanism to split each key to multiple key shares and routes the key shares to the holders on the fly through the key share routing paths. An example of key share routing layer is shown as Figure 5, which consists of the six holders determined by the node-joint multipath routing scheme (Figure 4) and two additional holders $H_{3,1}$ and $H_{3,2}$ to support the key share passing. In Figure 4, the paths only transmit the onion packages as the onion decryption keys, namely $K_1$, $K_2$ and $K_3$, have been pre-assigned to the holders at the start time $t_s$ and statically stored by the holders to wait for the arrival of the onion packages. However, in Figure 5, the key shares generated from the onion decryption keys through Shamir secret share scheme at $t_s$ are also transmitted through the paths together with the onion packages. Specifically, in Figure 5, at start time $t_s$, the sender node sends $K_1$ to holders $H_{1,1}$ and $H_{2,1}$ and an additional key $K_{3,1}$ to the holder $H_{3,1}$. Also, each of the three holders receives an onion package from the sender node. The two keys, $K_1$ and $K_{3,1}$, do not need to be split as there is no storage time for them. The three holders can thus decrypt the onion packages with the received onion decryption keys and obtain the IDs of the next three holders, namely $H_{1,2}$, $H_{2,2}$ and $H_{3,2}$, the three remaining onions and three key shares to be sent to the next holders at time $t_1$. Then, at $t_1$, each of $H_{1,2}$, $H_{2,2}$ and $H_{3,2}$ receives three onion packages and three key shares. The Shamir secret share scheme guarantees that the original key can be recovered from at least $m$ out of $n$ key shares. If we make $m = 2$ and $n = 3$ in this example, the onion decryption key can be recovered from the received key shares even if one key share is unavailable due to churn or attack. The holders $H_{1,2}$, $H_{2,2}$ and $H_{3,2}$ can restore onion decryption key $K_2$, $K_2$ and $K_{3,2}$ respectively and use them to decrypt the received onion packages to obtain the IDs of $H_{1,3}$ and $H_{2,3}$ and the onion packages and key shares to be sent to them. Similarly, at $t_2$, $H_{1,3}$ and $H_{2,3}$ restore the onion decryption key $K_3$ from the received key shares and decrypt the received onion package with $K_3$ to find out the secret key. Finally, at the release time $t_r$, the receiver node receives the secret key from $H_{1,3}$ and $H_{2,3}$. We can see that $K_1$, $K_2$ and $K_3$ have been sent to their target holders at time $t_s$, $t_1$ and $t_2$ respectively to arrive no earlier than the corresponding onion packages, as expected.

**Algorithm 1:** Key share routing scheme

```
Input : The replication factor $k$ and path length $l$ determined
        by the node-joint multipath routing scheme, the
        number of overall nodes available for path construction
        $N$, the expected emerging time $T$, exponential
        distribution parameter $\lambda$, node malicious rate $p$.
Output: Total share number $n$ and threshold share number $m$
        for each holder, the release-ahead attack resilience $R_r$
        and Drop attack resilience $R_d$.
1  $n = \lfloor \frac{N}{l} \rfloor$;
2  $p_{dead} = 1 - e^{-\frac{T}{\lambda l}}$;
3  $d = \lfloor p_{dead} * n \rfloor$;
4  $p_r = p_d = p$;
5  Initialize $P_r$, $P_d$ and $MN$;
6  Initialize $R_r = R_d = 1$;
7  for $column = 2$ to $l$ do
8      Calculate $m \in [1, n]$ that minimizes
         $Dif = |\sum_{i=m}^{n} \binom{n}{i} p^i (1-p)^{n-i} -$
         $\sum_{i=n-d-m+1}^{n-d} \binom{n-d}{i} p^i (1-p)^{n-d-i}|$;
9      $p_r = 1 - (1 - p_r)(1 - \sum_{i=m}^{n} \binom{n}{i} p^i (1-p)^{n-i})$;
10     $p_d = 1 - (1 - p_d)$
11            $(1 - \sum_{i=n-d-m+1}^{n-d} \binom{n-d}{i} p^i (1-p)^{n-d-i})$;
12     Add $p_r$ to $P_r$, $p_d$ to $P_d$ and $(m, n)$ to $MN$;
13 end
14 for $i = 0$ to $l - 1$ do
15     $R_r = R_r * (1 - (1 - P_r(i))^k)$;
16     $R_d = R_d * (1 - (P_d(i))^k)$;
17 end
18 $R_r = 1 - R_r$;
```

To complete the construction of key share routing paths, the values of $m$ and $n$ are required. The selection of $m$ and $n$ should consider both the attacks and the churn impact. We show the process to determine $m$ and $n$ and calculate attack resilience $R_r$ and $R_d$ as Algorithm 1. First, line 1 calculates the upper bound of the selection of $n$, where $N$ denotes the maximum number of nodes allowed to construct the key share routing paths and $l$ is the length of paths. In other words, we uniformly assign the resources (nodes) along the paths. Then, the number of dead nodes due to churn during holding time $t_h = \frac{T}{l}$ is estimated (line 2-3). As suggested by [2], the node death can be expressed by a decay pattern, namely the exponential mechanism $p_{dead} = 1 - e^{-\frac{1}{\lambda} t_h}$, where $\lambda$ denotes the average lifetime of nodes in the DHT. We can then estimate the dead shares during $t_h$ as $d$. We next start to calculate the $(m, n)$ and $(p_r, p_d)$ for each holder (line 4-13), where $p_r$ and $p_d$ refer to the release-ahead attack success rate and drop attack success rate respectively. The holders in the same column share same $(m, n)$ and $(p_r, p_d)$. For example, $H_{1,2}$, $H_{2,2}$ and $H_{3,2}$ in Figure 5 form the second column. The $p_r$ and $p_d$ for each column of holders are varying and therefore results in the requirement to adjust $m$ to handle the difference. Initially, the first column of holders have the smallest $p_r$ and $p_d$ (line 4). Then, the second column of holders have larger $p_r$ and $p_d$ because the adversary who fails to successfully attack the system at the first column can gain additional opportunity to either release or drop the secret key when enough key shares can be collected at second column. Therefore, the farther away from the beginning a column is, the larger $p_r$ and $p_d$ it will have. For each column from 2 to $l$, the adversary needs to gather $m$ out of $n$ shares to recover the onion decryption key to release the secret key or gather $n - d - m + 1$ out of $n - d$ alive shares to prevent the onion decryption key to be recovered. We can choose the value for $m$ that makes the difference between the success rates of the two attack objectives minimum (line 8) so that the system has both good release-ahead attack resilience and drop attack resilience without bias that makes it vulnerable. Once $m$ is determined, the $p_r$ and $p_d$ for that column can be updated (line 9-10) and all of them can be recorded (line 11). Finally, after the $p_r$ and $p_d$ for all the columns have been calculated, the attack resilience can be computed (line 14-18).

## IV. Experimental evaluation

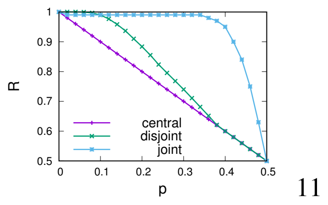
In this section, we experimentally evaluate the performance and security offered by the proposed scheme. Before reporting our results, we first briefly describe the experimental setup.

### A. Experimental setup

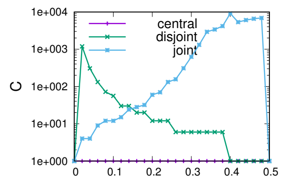
Our self-emerging key routing scheme is simulated through a Java-based DHT toolkit *Overlay Weaver* on an Intel Core i7 PC with 16GB RAM. We invoke 10000 DHT node instances and run each experiment for 1000 times to take the average. We randomly select $10000 * p$ non-repeated nodes and mark them as malicious. The probability density function of node death follows the exponential distribution suggested by [2].

### B. Experimental results

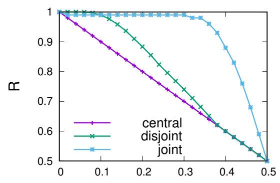
The experimental evaluation consists of three parts. First, we evaluate the release ahead attack resilience and drop attack resilience of the first three schemes, namely centralized scheme (*central*), node-disjoint multipath routing scheme (*disjoint*) and node-joint multipath routing scheme (*joint*). Then, we evaluate the impact of churn to the three pattern schemes as well as the key share routing scheme (*share*) and we show that the key share routing scheme has the best churn resilience. Finally, we evaluate the cost of key share routing scheme by adjusting the number of nodes available to construct the paths.



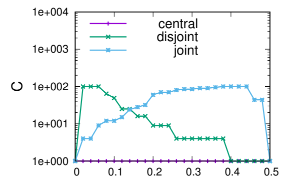
(a) attack resilience, 10000



(b) required nodes, 10000

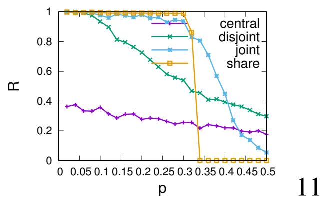


(c) attack resilience, 100

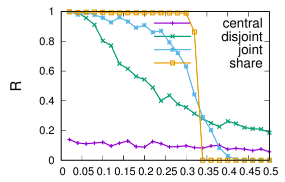


(d) required nodes, 100

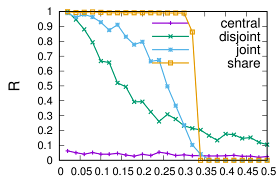
Fig. 6: Attack resilience evaluation



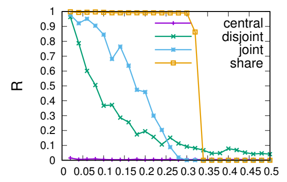
(a) $\alpha = 1$



(b) $\alpha = 2$



(c) $\alpha = 3$



(d) $\alpha = 5$

Fig. 7: Churn resilience evaluation

*1) Attack resilience evaluation:* In Figure 6(a), we evaluate the attack resilience $(R = R_r = R_d)$ of the three schemes with varying malicious note rate $p$ in 10000 nodes network. The number of nodes required to construct the routing paths is also measured and shown in Figure 6(b). The centralized scheme, which only requires a single node, works as the baseline. The node-disjoint multipath routing scheme makes $R > 0.9$ when $p <= 0.18$ but then rapidly drops to the baseline. Among the three schemes, the node-joint multipath routing scheme has the best performance in terms of attack resilience. It can keep $R > 0.99$ before $p = 0.34$ and $R > 0.9$ before $p = 0.42$. However, from Figure 6(b), we can see that the required nodes of node-joint multipath routing scheme, rapidly increases towards 10000 after $p = 0.15$ as the cost of high attack resilience increases. We then reduce the DHT network scale from 10000 to 100. Although Figure 6(d) shows that the required nodes of both node-disjoint multipath routing scheme and node-joint multipath routing scheme are suppressed due to the limited DHT nodes, we can see from Figure 6(c) that both the two schemes, especially the node-joint multipath routing scheme, still keeps good attack resilience. Therefore, we can conclude that the DHT scale do not influence the attack resilience significantly.

*2) Churn resilience evaluation:* The impact of churn is highly related to the value of expected emerging time $T$, namely the time duration during which the secret key stays in the DHT. Therefore, if we assume the average lifetime of a DHT node is $t_{life}$, we can evaluate the impact of churn by setting the emerging time $T$ to be $\alpha$ times of $t_{life}$. From Figure 7(a) to 7(d), we change $\alpha$ from 1 to 2, 3 and 5 to check the influence of churn with increasing $T$. We can observe that the centralized scheme is still the baseline of all. When $\alpha$ increases, the attack resilience of centralized, node-disjoint multipath routing and node-joint multipath routing schemes are reduced rapidly. In contrast, the key share routing scheme keeps nearly unchanged high attack resilience even for $\alpha = 5$ when $p < 0.3$. This means, if the average lifetime of a DHT node is one month, the key share routing scheme can successfully hide the secret key for 5 months and then release it if the adversary controls less than 30% DHT nodes. Such a long-term use case significantly increases the application range of the self-emerging data release system.

*3) Key share routing scheme cost evaluation:* Our last set of experiments evaluate the cost of key share routing scheme as the earlier experiments have demonstrated that the key share routing scheme offers best churn resilience, especially for large required emerging time $T$. However, the 10000 nodes used to form the path structure may be too costly for DHT networks with small or middle scale. Therefore, we want to evaluate the performance of key share routing scheme when the number of available nodes is smaller. Specifically, we change the number of available nodes from 10000 to 5000, 1000 and 100 and the results are shown in Figure 8. We set $\alpha = 3$. Obviously, the 10000-node cost shows the best result, which starts to drop from nearly $R > 0.99$ only after $p > 0.3$. The 5000-node cost shows very close performance, which only loses at $p = 0.32$. The 1000-node cost also offers good performance, which keeps $R > 0.95$ before $p > 0.26$. Finally, even the 100-node cost shows acceptable results, which keeps $R > 0.9$ before $p > 0.14$. Therefore, we can conclude that the cost can be significantly reduced by 10 times for most application cases and even by 100 times when $p$ is not large.

## V. Related work

Releasing private data to the future is a challenging research topic that has intrigued researchers for more than two decades. As its main technique, Timed-Release Encryption (TRE) was first proposed by May [12] in 1993. The TRE schemes can

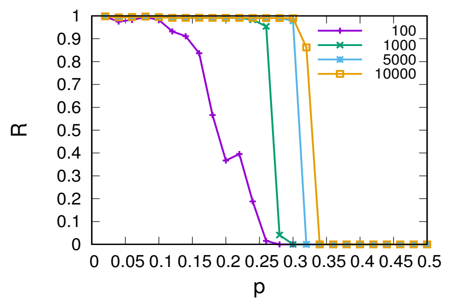


Fig. 8: Key share routing scheme cost evaluation

be mainly classified into two groups, namely time-lock puzzle scheme [1], [7], [16] and third party scheme [12], [16]. In 2003, [13] proposed a non-interactive TRE scheme based on quadratic residues (QR-TRE), but their time server has to be trusted. Later in 2005, bilinear pairing based TRE was proposed [3], but the time server is required to be curious. After that, although more efficient models and extensions were proposed [4], [6], [10], [23], most of them require the time server to be either trusted or curious, which becomes a bottleneck to the overall system security. Besides TRE, some efforts try to model the time by capturing 'real-world-time' [9], [19]. In [9], the role of time server can be eliminated by emulating the real-world time in a computational model based on Block-chain.

Another track of research efforts related to our work include self-destroying data represented by the Vanish system [8]. Vanish stores tiny bits of key information into a DHT network called Vuze. Research efforts [22] have shown that Vanish is vulnerable to both Sybil attack and hopping attack. Subsequently, the SafeVanish model has been proposed by [25] to increase the cost required to effectively attack the Vanish protocol. Vanish and its similar schemes [15], [24] take significant advantage of the decentralization and large scale features of distributed network to provide natural defense to attacks. In contrast to Vanish, our proposed work addresses the inverse of the problem addressed in the Vanish system to develop a DHT-based protocol for enabling self-emerging data. Our proposed schemes significantly leverage the large scale features of DHT networks to protect self-emerging data from being released prior to the release time.

## VI. Conclusion

In this paper, we propose new mechanisms for timed release of self emerging data using distributed hash tables. The proposed schemes leverage the use of large scale DHT networks to securely hide the protected data from being accessed prior to the release time. The proposed protocols enable automatic appearance of the stored encryption keys at the predetermined release time while keeping the protected encrypted data private and unavailable until the release time. We identity two key attacks in the proposed approach that could either make the hidden data appear before the prescribed release time (*release-ahead attack*) or destroy the hidden data altogether (*drop attack*). We present a suite of self-emerging key routing schemes on DHTs for securely storing and routing the encryption key in the DHT that achieve a high degree of resilience to both release-ahead and drop attacks. Our experimental evaluation using Overlay Weaver DHT emulator toolkit demonstrates the efficacy and attack-resilience of the proposed schemes.

## References

[1] Azar, Pablo, Shafi Goldwasser, and Sunoo Park. On time and order in multiparty computation. Cryptology ePrint Archive. Report 2015/178, 2015. http://eprint.iacr.org.

[2] Bhagwan, Ranjita, *et al*. Replication strategies for highly available peer-to-peer storage. Future directions in distributed computing, Springer Berlin Heidelberg, 153-158, 2003.

[3] Chan A C F, Blake I F. Scalable, Server-Passive, User-Anonymous Timed Release Public Key Encryption from Bilinear Pairing. IACR Cryptology ePrint Archive, 211, 2004.

[4] Cheon J H, Hopper N, Kim Y, *et al*. Provably secure timed-release public key encryption. ACM Transactions on Information and System Security (TISSEC), 11(2): 4, 2008.

[5] Douceur, John R. The sybil attack. International Workshop on Peer-to-Peer Systems, Springer Berlin Heidelberg, 2002.

[6] Emura K, Miyaji A, Omote K, *et al*. Time-specific encryption from forward-secure encryption. International Conference on Security and Cryptography for Networks. Springer Berlin Heidelberg, 184-204, 2012.

[7] Garay J A, Jakobsson M. Timed release of standard digital signatures. Financial Cryptography. Springer Berlin Heidelberg, 168-182, 2002.

[8] Geambasu R, Kohno T, Levy A A, *et al*. Vanish: Increasing Data Privacy with Self-Destructing Data. USENIX Security Symposium, 299-316, 2009.

[9] Jager, Tibor. How to Build Time-Lock Encryption. IACR Cryptology ePrint Archive, 478, 2015.

[10] Kikuchi R, Fujioka A, Okamoto Y, *et al*. Strong security notions for timed-release public-key encryption revisited. International Conference on Information Security and Cryptology. Springer Berlin Heidelberg, 88-108, 2011.

[11] Koufogiannis, Fragkiskos, Shuo Han, *et al*. Gradually Releasing Private Data under Differential Privacy, 2015.

[12] May T. Timed-release crypto. Unpublished manuscript, 1993.

[13] Mont M C, Harrison K, Sadler M. The HP time vault service: exploiting IBE for timed release of confidential information. Proceedings of the 12th international conference on World Wide Web, 160-169, 2003.

[14] Reed, Michael G., Paul F. Syverson, and David M. Goldschlag. Anonymous connections and onion routing. IEEE Journal on Selected areas in Communications, 16.4 (1998): 482-494.

[15] Reimann, Sirke, and Markus Drmuth. Timed revocation of user data: long expiration times from existing infrastructure. Proceedings of the 2012 ACM workshop on Privacy in the electronic society, ACM, 2012.

[16] Rivest R L, Shamir A, Wagner D A. Time-lock puzzles and timed-release crypto, 1996.

[17] Shamir, Adi. How to share a secret. Communications of the ACM, 22.11 (1979): 612-613.

[18] Singh, Atul. Eclipse attacks on overlay networks: Threats and defenses. In IEEE INFOCOM, 2006.

[19] Schwenk, Jrg. Modelling time for authenticated key exchange protocols. European Symposium on Research in Computer Security, Springer International Publishing, 2014.

[20] Stoica, Ion, *et al*. Chord: A scalable peer-to-peer lookup service for internet applications. SIGCOMM, ACM, 149-160, 2001.

[21] Stutzbach, Daniel, and Reza Rejaie. Understanding churn in peer-to-peer networks. SIGCOMM conference on Internet measurement, ACM, 2006.

[22] Wolchok S, Hofmann O S, Heninger N, *et al*. Defeating Vanish with Low-Cost Sybil Attacks Against Large DHTs. NDSS, 2010.

[23] Yavuz, Attila Altay, and Peng Ning. Self-sustaining, efficient and forward-secure cryptographic constructions for Unattended Wireless Sensor Networks. Ad Hoc Networks, 10.7: 1204-1220, 2012.

[24] Zarras, Apostolis, *et al*. Neuralyzer: flexible expiration times for the revocation of online data. Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy, ACM, 2016.

[25] Zeng L, Shi Z, Xu S, *et al*. Safevanish: An improved data self-destruction for protecting data privacy. CloudCom, 521-528, 2010.