Emerge: Self-emerging Data Release using Cloud Data Storage

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*—In the age of Big Data, advances in distributed technologies and cloud storage services provide highly efficient and cost-effective solutions to large scale data storage and management. Supporting self-emerging data using clouds is a challenging problem. While straight-forward centralized approaches provide a basic solution to the problem, unfortunately they are limited to a single point of trust. Supporting attack-resilient timed release of encrypted data stored in clouds requires new mechanisms for self emergence of data encryption keys that enables encrypted data to become accessible at a future point in time. Prior to the release time, the encryption key remains undiscovered and unavailable in a secure distributed system, making the private data unavailable. In this paper, we propose *Emerge*, a self-emerging timed data release protocol for securely hiding data encryption keys of private encrypted data in a large-scale Distributed Hash Table (DHT) network that makes the data available and accessible only at the defined release time. We develop a suite of erasure-coding-based routing path construction schemes for securely storing and routing encryption keys in DHT networks that protect an adversary from inferring the encryption key prior to the release time (*release-ahead attack*) or from destroying the key altogether (*drop attack*). Through extensive experimental evaluation, we demonstrate that the proposed schemes are resilient to both *release-ahead attack* and *drop attack* as well as to attacks that arise due to traditional churn issues in DHT networks.**

I. Introduction

With the sheer size of data generated in the age of Big Data, fueled by the rapid development of cloud and Big Data technologies, we are witnessing an extensive use of cloud storage services for efficient and cost-effective management of large scale data. Supporting self-emerging data using clouds is a challenging problem. While straight-forward centralized approaches provide a basic solution to the problem, unfortunately they are limited to a single point of trust.

Several scenarios create a need for self emerging data. For example, non-releasable private data may become releasable due to the degradation of time-varying data privacy [10]. Other examples include personal data of individuals (e.g., medical diagnostics information, web browsing patterns, location trajectory patterns) collected during the childhood and adult life of individuals. The sensitivity of such data may drop as the individual ages and after the end of the individual's life, requiring self-emergence of the protected data. The timed-release mechanism can also help audit certain time-sensitive online events. For instance, in a secure voting mechanism, the encrypted vote of individuals may be collected during the polling process and the collected 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.

Enabling self-emerging data release in clouds requires mechanisms to guarantee that the encrypted data stored in clouds are decrypted only at the release time, thus making the raw private data *self-emerge*. For this purpose, the encryption key, which is generated at the time of encryption, has to be securely protected and delivered to the legitimate receivers at the release time. In a naive centralized approach, the encryption key can be secretly stored on a trusted third party server until the release time, but this simple strategy may fail to protect adequate privacy because of the single point of trust. Specifically, the adversary can locate the key and focus on breaching the security through insider attacks or the service provider may be forced to release the key prior to the release time which can violate the intended privacy provided by self-emerging data.

In this paper, we present *Emerge*, a self-emerging timed data release protocol for securely hiding encryption keys of encrypted data in large-scale Distributed Hash Table (DHT) [19] networks. In *Emerge*, the encryption key is packaged, split into multiple fragments through erasure coding [21] and routed among the DHT nodes along a pre-determined pseudorandom routing path pattern to the receiver so that the key can be recovered exactly at the release time by the receiver.

The proposed *Emerge* protocol ensures private data to be securely hidden and inaccessible until the release time and enables self emergence of it at the release time. We note that the objectives of the protocol can be significantly challenged by adversaries controlling a sizable proportion of the DHT network. When a sufficient number of DHT nodes are compromised by an adversary, s/he can either release the hidden data before the prescribed release time (*release-ahead attack*) or destroy the hidden data altogether (*drop attack*). These two specific attacks in combination with traditional churn issues [20] in DHTs constitute significant challenges to the design of the *Emerge* protocol. Thus, ensuring high resilience to churn and to release-ahead attacks and drop attacks is a

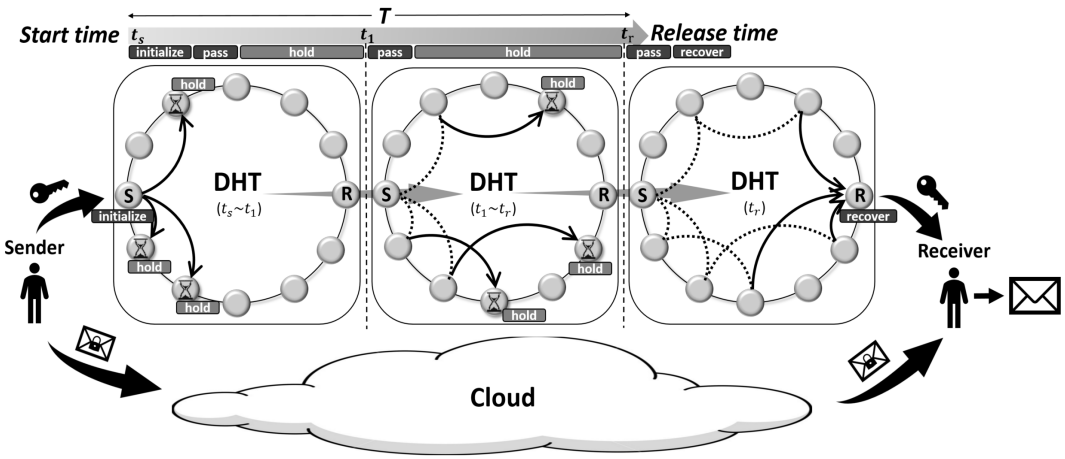


Fig. 1: Self-emerging cloud data storage system

central objective of the system design. We conduct extensive experimental evaluation of the proposed protocol using Overlay Weaver DHT toolkit [17] and the results demonstrate that the proposed erasure-coding-based schemes provide high resilience to both release ahead and drop attacks and as well as to churn issues in DHT.

The rest of the paper is organized as follows. We present the system model that supports the *Emerge* protocol and discuss the potential attack and churn issues in Section II. We then explain the proposed *Emerge* protocols in detail in Section III. We evaluate the proposed techniques through extensive experiments in Section IV. Finally, the related work is presented in Section V and we conclude in Section VI.

II. System and models

In this section, we first present the self-emerging cloud data storage system, and then discuss the main challenges, including the adversary models and churn.

A. Self-emerging cloud data storage system

There are four major entities in the self-emerging cloud data storage system, namely the data senders, the data receivers, the DHT network and the cloud, as shown in Figure 1. As the owner of the data, the data senders want to protect the data stored in the cloud from being accessed until a pre-defined data release time. The data however needs to be accessible to the receivers after the release time. The data senders encrypt their data before uploading to the cloud and they use the DHT network to make their encryption keys 'disappear' before the future data release time. The encryption key automatically appears to the receivers at the release time, allowing the receivers to decrypt the protected data.

If we denote the start time as t_s and the expected future release time as t_r, we can express the entire time period that the data owner wants to make the encryption key disappear as T. It is referred to as the emerging time period as the encryption key is emerging from the DHT to the intended receiver during T. At start time t_s, the sender encrypts her data and sends the encrypted data and encryption key to the cloud and the DHT network respectively (shown as 'initialize' in Figure 1). During the emerging time period T, the encryption key will be packaged, split to multiple packages and routed among the nodes in the DHT. Each package, after being stored by a node in the DHT for a limited time period (shown as 'hold' in Figure 1), will be sent to the next node and gets routed towards the receiver along a carefully designed path. At the release time t_r, the receiver can collect the packages from the DHT network to recover the encryption key (show as 'recover'). She can then download the encrypted data from the storage cloud and decrypt the data using the key. It is easy to see that the DHT network takes the core role in the system for the *Emerge* protocol. Next, we analyze the challenges and attacks that lead to the compromise of the stored key in the DHT network.

B. Adversary models

Based on the objective of the adversary, we define two attack models, namely the release-ahead attack aiming to extract the encryption key from DHT network before the release time t_r and the drop attack aiming to prevent the encryption key to be recovered by the legitimate receiver after t_r. For both the attack models, the adversary needs to control a fraction of DHT nodes which can be realized through Sybil attack [5], Eclipse attack [18] or collusion with other adversaries.

In release ahead attack, the adversary, which may include the legitimate receiver, aims to extract the encryption key from the DHT network to decrypt the data before the release time t_r. They can collect the encryption key packages from their controlled DHT nodes if the packages pass these malicious nodes. This attack is effective in many scenarios that require timed release of self-emerging data. For instance, in an online exam scenario, if the adversary can decrypt the exam questions before other participants, he can earn more time to answer the questions and affect the fairness of the exam.

In drop attack, the adversary aims to prevent the encryption key to be received by the legitimate receiver to decrypt the data in the cloud at release time t_r. The adversary can drop the encryption key packages from their controlled DHT nodes if the packages pass these malicious nodes. A successful drop attack can make the encrypted data still inaccessible after the

release time t_r. In those time-sensitive scenarios, this means the scheduled activity has to be canceled. For example, the online exam cannot be started. In addition, since the key for the encrypted data is lost, unless the adversary releases those dropped packages again, the encrypted data can never be decrypted.

C. Churn impact

The data storage in DHT networks always suffers from the churn issue [20]. The churn impact to our system can be summarized as short-term and long-term impacts. The short-term impact is caused by the transiently left nodes. The nodes leave the network for a short time period, so their ID and responsibilities have not been transferred to other nodes. This may block the routing of encryption key packages in the network for a short time period but its effect is limited. However, the long-term impact is vital to our system. A node is 'dead' when it leaves the network for a long time and its ID and responsibility in the encryption key package routing is deprived. Because of the death of the nodes, the encryption key packages stored on them are also lost. Even if the packages are replicated to other nodes, we note that the new nodes also have probability p (the fraction of nodes controlled by the adversary) to be malicious, thus significantly increasing the chance for the adversary to get it and run release-ahead attack successfully. For example, in the case that a package is replicated to three nodes and two of them are dead and then replaced by new nodes during the emerging time period T, the probability for the adversary to get the package increases from $1-(1-p)^3$ to $1-(1-p)^5$, where p stands for the fraction of nodes in DHT controlled by the adversary. Therefore, the conventional replication [21] is not adopted in this paper.

Motivated by those challenges, we propose the *Emerge* protocols for the DHT network to carefully package and route the encryption key during the emerging time period T to handle both the attack models and churn.

III. Emerge protocols

In this section, we present the proposed *Emerge* protocols in detail. The *Emerge* protocol consists of three components, namely routing path construction, initial package generation and package routing, to be implemented in a sequential order. Specifically, at start time t_s, the sender enters DHT network as a node. It first locally determines routing path pattern based on the adopted pattern construction scheme and pseudo-randomly select DHT IDs to fill in the pattern. Then, based on the pattern and selected IDs, the sender node locally generates the initial data packages. Finally, it sends the initial packages out and the packages will be routed and processed along the paths to deliver the encryption key to the receiver at the release time t_r.

Based on the adopted routing path construction schemes, we propose three *Emerge* protocols and all of them are based on the erasure coding mechanism [21]. As a common mechanism to protect data, erasure coding [21] can divide a data package to m fragments and recode them into n fragments so that the package can be recovered from any m fragments ($m \leq n$). We start from the one-hop path pattern scheme, which applies erasure coding to establish multiple one-hope paths between sender and receiver to guarantee attack resilience. Then, by taking dead nodes (churn) into account, we propose the adjusted one-hop path pattern scheme to make it resilient to churn issues by estimating the number of dead nodes and adjusting the parameters of erasure coding based on the estimation. After that, by dividing the entire emerging time period $T = t_s - t_r$ into several shorter time periods, we propose the multi-hop path pattern scheme to iteratively implement the erasure coding mechanism to route the packages so that the lose of packages during each shorter time period can be suppressed and the lost packages can be recovered by multiple usages of erasure coding. To compare the schemes in terms of attack resilience, we measure the release-ahead attack resilience, R_r as the probability that an adversary fails to restore the encryption key package before the legitimate release time t_r, and drop attack resilience, R_d as the probability that an adversary fails to prevent the encryption key package to be restored by the receiver at the release time t_r. In this paper, we desire $R_r = R_d$ because we expect that the proposed *Emerge* protocol has both good release-ahead attack resilience and good drop attack resilience without compromising either of them and making the protocol vulnerable.

A. One-hop scheme

The one-hop path pattern scheme applies the erasure coding to split the encryption key package into n fragments and send each of them through an one-hop path to receiver to allow at most $n-m$ of the fragment transmissions to be unsuccessful. In other words, n holder nodes are applied to store the n fragments for the entire emerging time period T. We name all the nodes selected by the sender to form the path pattern as holder nodes. Each holder node receives package(s) from its predecessor(s), holds(stores) the packages for a time period (in this scheme the entire emerging time period T) and sends the packages to its successor(s) after that time period.

1) Routing path construction: To construct the one-hop path pattern, we need to determine the total number of fragments, n and the minimum (threshold) number of fragments to restore the encryption key package, m. Given the maximum available nodes that can be used to form the pattern (namely the limited recourse), N, we can calculate the value of n and m to maximize the attack resilience through Algorithm 1.

Algorithm 1: One-hop path pattern

Input : Maximum available node number N.
Output: Total fragment number n, threshold fragment number m.

$m = \lfloor \frac{N+1}{2} \rfloor$;
$n = 2m - 1$;

If an adversary controls at least m holder nodes, the encryption key package will be restored at start time and the release-

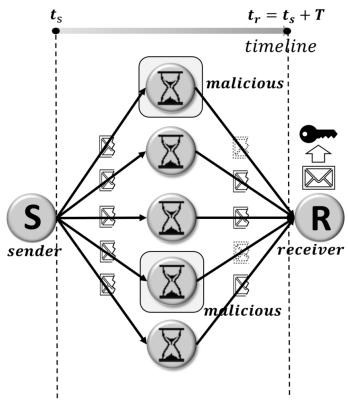


Fig. 2: One-hop scheme

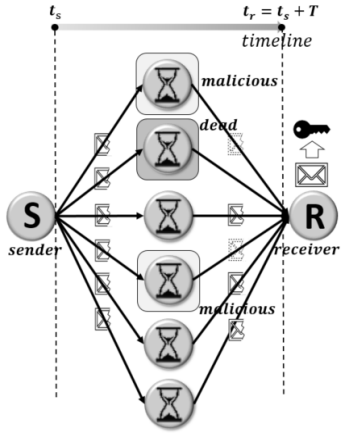


Fig. 3: Adjusted one-hop scheme

ahead attack is successful. If the adversary controls at least $n-m+1$ holder nodes, the receiver will fail to receive at least m fragment to restore the encryption key package at release time t_r, which makes drop attack successful. Therefore, by setting $n-m+1 = m$, namely $n = 2m-1$, we get equivalent release-ahead attack resilience R_r and drop attack resilience R_d.

Lemma 1. *In one-hop scheme, with $R_r = R_d$, a larger n makes attack resilience R_r and R_d higher.*

Proof. From [15], we can get:

$$\frac{n}{m} = \left(\frac{\sigma\sqrt{\frac{p(1-p)}{m}} + \sqrt{\frac{\sigma^2 p(1-p)}{m} + 4(1-p)}}{2(1-p)}\right)^2 \quad (1)$$

where p denotes the probability that a random DHT node is malicious and σ is positively proportional to the R_d. To get $R_r = R_d$ so that the system has good attack resilience towards both the two attacks, we need $n = 2m-1$, namely $\frac{n}{m} = \frac{2m-1}{m} \approx 2$. Therefore, in equation 1, with fixed value of $\frac{n}{m}$ and p, a larger m makes σ larger and therefore R_d higher. Since $n \approx 2m$ and $R_r = R_d$, we can conclude that lager n makes R_r and R_d higher. □

Therefore, given the limited available nodes N to form the pattern, we need to maximize n to maximize the attack resilience, so we set $m = \lfloor \frac{N+1}{2} \rfloor$ (line 1) to maximize m as an integer and then get $n = 2m-1$ (line 2).

2) Initial package generation: The sender generates n initial data packages as the n fragments of encryption key package.

3) Package routing: At start time t_s, the sender node sends the n initial data packages to the n holder nodes. The n holder nodes hold the packages for the entire emerging time period T. At the release time $t_r = t_s + T$, the holder nodes send the packages to the receiver node.

4) Security analysis: The release-ahead attack resilience is $R_r = 1 - \sum_{i=m}^{n} \binom{n}{i} p^i (1-p)^{n-i}$ and drop attack resilience is $R_d = 1 - \sum_{i=n-m+1}^{n} \binom{n}{i} p^i (1-p)^{n-i}$, as the success rate of both the attack models follows binomial distribution. In the example shown in Figure 2 with $n = 5$ and $m = 3$, two holder nodes are malicious. In release-ahead attack, the two malicious holder nodes can get two fragments, which is less than m to restore the encryption key package. In drop attack, the two malicious holder nodes can drop two fragments, but the receiver can still get $3 = m$ fragments to restore the encryption key package at release time t_r.

B. Adjusted one-hop scheme

If the emerging time period T is long, more holder nodes may become dead, which makes their stored fragments get lost. We can approach this problem through two solutions. The first solution is to generate a new fragment when one fragment is lost. However, to generate the new fragment, at least m living fragments have to be collected by one DHT node to restore the encryption key package and re-generate the n fragments through erasure coding. This means the encryption key package has to be known by one node, which significantly increases the success rate of release-ahead attack because this node has probability p to be malicious. In this paper, we apply our second solution. We estimate the number of dead holder nodes as d and reserve some fragments for them through adjusting m and n.

Algorithm 2: Adjusted one-hop path pattern

Input : Maximum available node number N, emerging time period T.
Output: Total fragment number n, threshold fragment number m.

1 $n = N$;
2 $p_{dead} = 1 - e^{-\frac{1}{\lambda}T}$;
3 $d = p_{dead} * n$;
4 **for** $m = 1$ *to* n **do**
5 $\quad Dif = |\sum_{i=m}^{n} \binom{n}{i} p^i (1-p)^{n-i} - \sum_{i=n-d-m+1}^{n-d} \binom{n-d}{i} p^i (1-p)^{n-d-i}|$
6 **end**
7 $m =$ the value between 1 and n that minimizes Dif.

1) Routing path construction: As suggested by [1], the node death in DHT network can be expressed by a decay pattern, namely the exponential distribution. We can then estimate the percentage of dead holder nodes after the emerging time period T to be $p_{dead} = 1 - e^{-\frac{1}{\lambda}T}$, where λ is the average DHT node lifetime. (In [1], λ is set to 3 years, but it can be changed for different DHT networks.) Therefore, among the n total holder nodes, the number of dead holder nodes should be $d = p_{dead} * n$, which makes the number of living holder nodes to be $n - d$. To do drop attack, the adversary should drop at least $n - d - m + 1$ fragments among the $n - d$ living fragments to make the receiver can at most get $m - 1$ fragment at the release time t_r and fails to recover the encryption key package. The probability for this, which follows the binomial distribution, can be calculated by $P_d = \sum_{i=n-d-m+1}^{n-d} \binom{n-d}{i} p^i (1-p)^{n-d-i}$. However, the dead nodes have no influence to the release-ahead attack because the adversary can collect the fragments at the beginning of emerge process before any node death. The probability for the adversary to collect at least m fragments from the

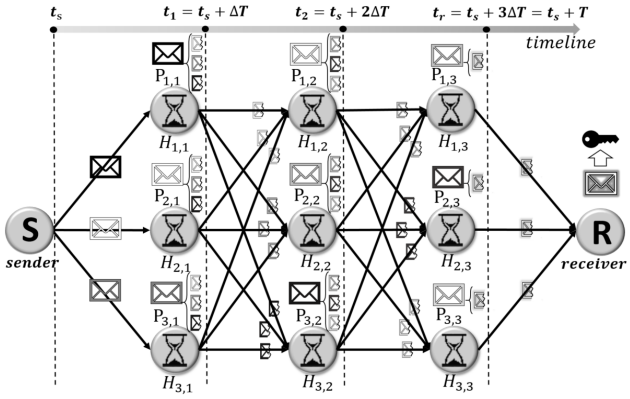


Fig. 4: Multi-hop scheme

total n fragments to restore the encryption key package is $P_r = \sum_{i=m}^{n} \binom{n}{i} p^i (1-p)^{n-i}$. Given the maximum available nodes N and emerging time period T, we can calculate m and n through Algorithm 2. We first set $n = N$ (line 1) to maximize attack resilience(the proof is similar to that of lemma 1), and then estimate the dead nodes (line 2-3). After that, we try to find the value of m between 1 and n to make $P_d = P_r$ so that $R_d = R_r$ (line 4-7).

2) Initial package generation: We use the same approach as the one-hope scheme.

3) Package routing: We use the same approach as the one-hope scheme.

4) Security analysis: In the example shown as Figure 3, we have maximum available nodes $N = 6$ and estimate $p_{dead} = \frac{1}{6}$. Therefore, we can calculate $n = N = 6$, $d = 1$ we assume we get $m = 3$ from Algorithm 2. If there are two malicious holder nodes and one dead holder nodes after the emerging time period T, the adversary will fail to restore the encryption key package with $2 < m$ fragments and the receiver can successfully recover the encryption key package with $3 = m$ received fragments.

Although the adjusted one-hop scheme is resilient to both attack resilience and churn, it has two security problems due to large emerging time period T. To better express T in time scale, we represent T to be α times of average DHT node lifetime, λ. We believe this is reasonable because different DHT networks may have different average DHT node lifetime. First, when emerging time period T is large, the percentage (p_{dead}) of nodes to be dead may be quite large. For example, if we set $p_{dead} = 0.99 = 1 - e^{-\frac{1}{\lambda}T}$, we can get $\alpha = 4.6$, which means 99% fragments will be lost due to churn after 4.6 times of average DHT node lifetime, λ. In such cases, the adjusted one-hop scheme fails to make attack resilience high. Another issue in this context is the dead node error. An implicit assumption for the scheme is the accuracy of the estimation. Unfortunately, in practice, the number of dead nodes does not always match with the estimation, which causes errors.

C. Multi-hop scheme

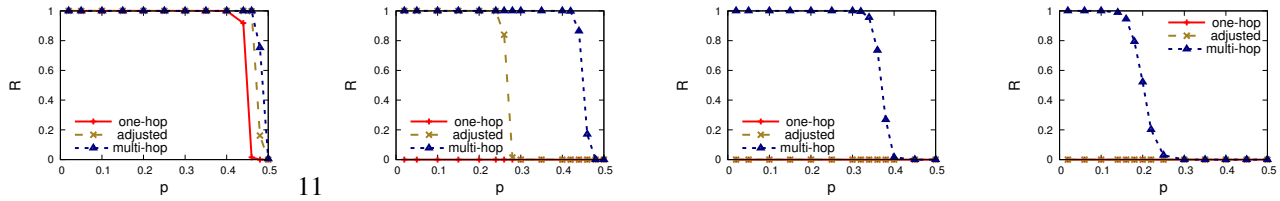
To solve the security problems due to long emerging time period T in one-hop schemes, we propose the multi-hop scheme. Instead of deploying a single set of nodes (Figure 3) to hold the packages during the entire T, we now arrange multiple sets of nodes (Figure 4) to carry the packages in relay from the sender to the receiver. Also, the single usage of the erasure coding is now extended to a nested usage so that the old packages can be merged at each set of nodes to generate new packages and the reduced number of alive packages can be replenished during each re-generation. Specifically, we divide the entire long T to l pieces of short time periods, namely $T = l * \triangle T$. To form the path pattern, we need l sets of nodes to carry the packages in relay and each set to take charge of the packages for $\triangle T$, namely $(i-1)\triangle T \leq t < i\triangle T$, where $i \in [1, l]$. Each set contains n nodes, so the entire number of nodes to form the path pattern is $n*l$, which should be pseudo-randomly selected by the sender node in a non-repeated way. At the start of the i^{th} short time period, namely the time $t = (i-1)\triangle T$, each node in the i^{th} set receives one package from each node in the $(i-1)^{th}$ set. Ideally, the number of received packages should be n. However, some packages may be lost due to drop attack (the $(i-1)^{th}$ set has malicious nodes) or churn (some nodes in the $(i-1)^{th}$ set become dead during the $(i-1)^{th}$ short time period). If the number of received packages is at least m, the node in the i^{th} set can still successfully merge the received packages to get the one generating them through erasure coding (called parent package). This parent package consists of the n new packages and the IDs of the nodes in the $(i+1)^{th}$ group. At the end of the i^{th} short time period, namely the time $t = i\triangle T$, the n new packages are sent to the n nodes in the $(i+1)^{th}$ group. The whole process is then repeated during the next short time period $\triangle T$. This routing scheme has two advantages. First, since $\triangle T < T$, the lost packages during $\triangle T$ is much fewer than T. Second, by iteratively implementing erasure coding, each group of n nodes can recover the lost packages caused by drop attack or churn during the previous short time period.

1) Routing path construction: Besides the values of m and n for erasure coding, the multi-hop scheme also needs

Algorithm 3: Multi-hop path pattern

Input : Maximum available node number N, emerging time period T, DHT node average lifetime λ.
Output: Total fragment number n, threshold fragment number m, number of groups of n holder nodes l.

1 **for** $l = 1$ *to* $\lfloor 5(\frac{T}{\lambda}+1)2^{\lg N-3} \rfloor$ **do**
2 $\quad n = \lfloor \frac{N}{l} \rfloor$;
3 $\quad p_{dead} = 1 - e^{-\frac{T}{\lambda l}}$;
4 $\quad d = p_{dead} * n$;
5 $\quad$ **for** $m = 1$ *to* n **do**
6 $\quad\quad Dif = |\sum_{i=m}^{n} \binom{n}{i} p^i (1-p)^{n-i} - \sum_{i=n-d-m+1}^{n-d} \binom{n-d}{i} p^i (1-p)^{n-d-i}|$
7 $\quad$ **end**
8 $\quad m =$ the value between 1 and n that minimizes Dif;
9 $\quad R_r = (1 - \sum_{i=m}^{n} \binom{n}{i} p^i (1-p)^{n-i})^l$;
10 $\quad R_d = (1 - \sum_{i=n-d-m+1}^{n-d} \binom{n-d}{i} p^i (1-p)^{n-d-i})^l$;
11 **end**
12 The selected (l, n, m) maximizes $min(R_r, R_d)$.



(a) $T = 0.1\lambda$ (b) $T = \lambda$ (c) $T = 10\lambda$ (d) $T = 100\lambda$

Fig. 5: Varying emerging time period T

to decide the value of l, namely the number of short time period $\triangle T = \frac{T}{l}$, also as the number of sets of nodes. The multi-hop path pattern algorithm is shown as Algorithm 3. Assume we have determined l, since the number of maximum available nodes is $N = n * l$, we can get $n = \lfloor \frac{N}{l} \rfloor$ (line 2). Then, we estimate the dead nodes d during the short time period $\triangle = \frac{T}{l}$ (line 3-4) and find the value for m (line 5-8, same as Algorithm 2). We can then calculate the attack resilience R_r and R_d for each value of l. Our objective is to find the value of l that maximize $min(R_r, R_d)$ from the range $[1\ ,\ \lfloor 5(\frac{T}{\lambda}+1)2^{\lg N-3}\rfloor]$, where the upper bound of the range is estimated through simulation in Section IV.

2) Initial package generation: The sender node should first run algorithm 3 to determine l, n, m and pseudo-randomly choose non-repeated IDs for the selected nodes. After that, the sender node can pretend to be the receiver node, which just recovers the final package containing the encryption key. Then, the sender node can split the final package into n packages through erasure coding and assume these n packages are the parent packages maintained by the nodes in the last set (l^{th} set), called l^{th} parent packages. Next, the sender node should split each of the l^{th} parent package into another n packages received by the node in the 1^{st} set from the n nodes in the $(l-1)^{th}$ set. At this stage, the sender node should have n^2 packages because there are n nodes in the $(l-1)^{th}$ set and each of them sends n packages to the l^{th} set of nodes (as shown in Figure 4). The n packages for each node in the $(l-1)^{th}$ set can be merged to generate the parent package maintained by it so that the sender only need to keep the $(l-1)^{th}$ parent packages to save space. By repeating this, the sender node can get the $(l-2)^{th}$ parent packages, $(l-3)^{th}$ parent packages... and finally the 1^{st} parent packages maintained by the 1^{st} set of nodes, which are actually the initial packages sent from the sender node to them.

3) Package routing: Figure 4 shows an example of multi-hop scheme with $l = 3$ and $n = 3$. At start time t_s, the sender node sends three initial packages to the three 1^{st} group nodes. Each 1^{st} group node gets three contained packages from the initial package, stores them for a short time period $\triangle T$ and send the three packages to the three 2^{nd} group nodes at $t_1 = t_s + \triangle T$. Then, each 2^{nd} group node restores a package from the received packages, gets three contained packages, holds them for $\triangle T$ and sends them to the three 3^{rd} group nodes at $t_2 = t_s + 2\triangle T$. Finally, each 3^{rd} group node restores a package from the received packages, gets the encryption key package fragment, holds it for $\triangle T$ and sends it to the receiver node at t_r to make the receiver restore the encryption key package.

4) Security analysis: The multi-hop scheme has better performance when the emerging time T is large. When the Algorithm 3 gives $l = 1$, the output pattern is same as the one generated by the adjusted one-hop scheme. Therefore, we can consider the adjusted one-hop scheme to be a special case of multiple-hop scheme.

IV. Experimental evaluation

In this section, we experimentally evaluate the performance and security offered by the proposed *Emerge* protocols. Before reporting our results, we first present our experimental setup.

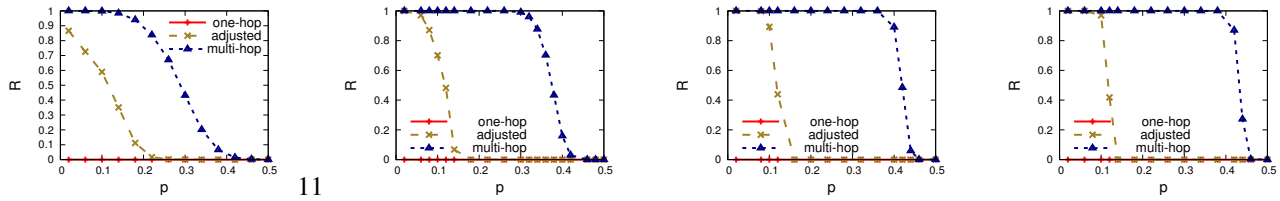
A. Experimental setup

We use an Intel Core i7 PC with 16GB RAM to simulate the *Emerge* protocols through the Java-based DHT toolkit *Overlay Weaver*. We invoke at most 10000 DHT node instances and repeat each experiment for 1000 times to show the average results. To evaluate the attack resilience, we mark a DHT node as malicious with probability p. To evaluate the churn resilience, we set a lifetime for each DHT node, which follows exponential distribution suggested by [1].

B. Experimental results

In our experiments, we first evaluate the impact of varying emerging time period T to the performance and security of the three *Emerge* protocols with one-hop scheme (one-hop), adjusted one-hop scheme (adjusted) and multi-hop scheme (multi-hop) respectively. Then, we evaluate the impact of the maximum available nodes N, namely the limited resource to construct the path pattern, to the *Emerge* protocols. Finally, we discuss the selection of the upper bound of l range in Algorithm 3.

The first set of experiments evaluates the *Emerge* protocols with varying emerging time period T. The objective of *Emerge* protocol is to hold and hide the encryption key in the DHT network for the emerging time period T. Therefore, the value of T is an important factor to evaluate it. If the *Emerge* protocol can only effectively work for short T, we do not find its performance to be good enough to satisfy long emerging times. A longer emerging time period T may result in more dead nodes, which requires the *Emerge* protocol to be both churn-resilient and attack-resilient. To comprehensively understand the performance of the protocols, we measure their attack resilience with four representative value of T, namely short emerging time period $T = 0.1\lambda$, medium emerging



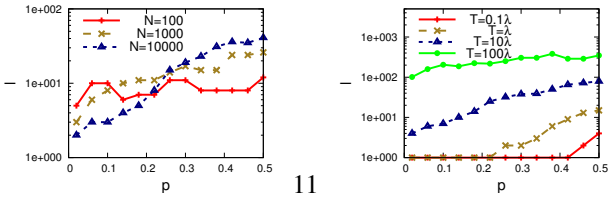
(a) $N = 100$ (b) $N = 1000$ (c) $N = 5000$ (d) $N = 10000$

Fig. 6: Varying path construction resource N

time period $T = \lambda$, long emerging time period $T = 10\lambda$ and very long time period $T = 100\lambda$, where λ denotes the average lifetime of DHT nodes (e.g. three years in [1]). Since we equally treat the release ahead attack resilience R_r and drop attack resilience R_d, the measured $R = R_r = R_d$. The maximum available nodes N is fixed to 10000. In Figure 5(a), we measure attack resilience R with varying p for the short emerging time period $T = 0.1\lambda$. All the three protocols show good R. Even the protocol based on one-hop scheme, which is most susceptible to T, can maintain quite high R when $p \leq 0.44$. However, even for short T with little churn impact, we can see the performance of multi-hop scheme is the best. In Figure 5(b), the emerging time period is 10 times than the previous one, which makes the churn impact start to be strong. For the one-hop and adjusted one-hop schemes, since $p_{dead} = 1 - e^{-1} = 0.63$, 63% of the n fragments has been lost due to churn. As can be seen, since the one-hop does not adjust n and m for this, its R directly drops to 0. In contrast, the adjusted one-hop scheme adjusts the value of m by estimating the number of dead nodes and its R is still good before $p = 0.26$. Compared with these two schemes, the performance of the multi-hop scheme is much better. The reason for that is the partitioning of the entire emerging time period T. By dividing it to multiple small pieces, the number of dead nodes during each small time period can be reduced and the lost fragments can also be recovered by the iterative erasure coding. In Figure 5(c), the emerging time period T is further increased by 10 times. Such a long T makes nearly 100% nodes to be dead for the two one-hop schemes and results in their $R = 0$. In contrast, although the multi-hop scheme is also affected, it still keeps $R > 0.99$ when $p \leq 0.32$. Finally, even for the very long T in Figure 5(d), the multi-hop scheme can still maintain $R > 0.99$ when $p \leq 0.14$. As can be seen, all the three schemes work well for short T. The adjusted one-hop scheme can keep good performance for medium T, but only the multi-hop scheme can work well even for long and very long T.

The second set of experiments evaluates *Emerge* protocols with maximum total available node N to build the path pattern, namely the path construction resource. Our default choice of N is 10000, which means the routing path pattern is constructed by 10000 DHT nodes. This is acceptable for large-scale DHT network. However, in practice, if a DHT network is not big enough to support $N = 10000$, we want to understand the impact of reduced N to the performance of *Emerge* protocols by reducing N to 5000, 1000, and 100. For this set of experiments, we set the emerging time period $T = 2\lambda$, which has made the attack resilience of the one-hop attack drop to 0 even for $N = 10000$, so we mainly focus on the performance of adjusted one-hop scheme and multi-hop scheme. In Figure 6(a), N is reduced to 100, which means the routing path pattern is formed by at most 100 DHT nodes. We can find that the attack resilience R of one-hop scheme rapidly drops from 0.866 for $p = 0.02$ to 0.422 for $p = 0.12$. In contrast, the attack resilience R of multi-hop scheme keeps higher than 0.99 before $p = 0.14$ and drops lower than 0.5 when p is around 0.29. We can conclude that the multi-hop scheme can still effectively work for small path pattern with $N = 100$ when the probability of node to be malicious p is not high but the adjusted one-hop scheme does not work well. In Figure 6(b), we increase N by 10 times. Both the two schemes have improved attack resilience. Specifically, the adjusted one-hop scheme can make $R > 0.9$ for $p < 0.08$ and $R > 0.5$ for $p < 0.13$, and the multi-hop scheme can make $R > 0.99$ for $p < 0.30$ and $R > 0.5$ for $p < 0.38$. Then, we further increase N by 5 times to 5000 in Figure 6(c). The adjusted one-hop scheme can make $R > 0.9$ for $p < 0.10$ and $R > 0.5$ for $p < 0.12$, and the multi-hop scheme can make $R > 0.99$ for $p < 0.38$ and $R > 0.5$ for $p < 0.42$. We find that the reduction way of the attack resilience R of the adjusted one-hop scheme along the increasing p changes from a smooth manner to a steep manner from $N = 1000$ to $N = 5000$. We can consider $p = 0.12$ as the threshold. When N is small, the R value for $p < 0.12$ gradually drops from 1 to 0.5 and the R for $p > 0.12$ gradually drops from 0.5 to 0. In contrast, when N is large, the R keeps close to 1 for $p < 0.12$ and suddenly drops to almost 0 after $p = 0.12$. In Figure 6(d), the attack resilience of adjusted one-hop scheme has little change. The multi-hop scheme slightly increases $R > 0.99$ for $p < 0.40$. As can be seen, the value of N can be reduced to 5000 from 10000 without losing big performance.

The goal of the third set of experiments is to reasonably bound the selection of l in the Algorithm 3 because we have proved the multi-hop scheme is the most effective one and the unnecessary loops in l selection can drop the performance of Algorithm 3. In Figure 7(a), we fix $T = 2\lambda$ and measure value of l with varying p for $N = 100, 1000, 10000$. We can see that the upper bound of l happens at large p and changes from 12 to 26 to 41 for $N = 10^2, 10^3, 10^4$ respectively so that we can roughly summarize the increment to be twice when N increases from 10^i to 10^{i+1}. In Figure 7(b), we fix $N = 10000$ and measure value of l with varying p for



(a) Varying N (b) Varying T

Fig. 7: l upper bound selection

$T = 0.1\lambda, \lambda, 10\lambda, 100\lambda$. We can find the upper bound of l also happens at large p and can be bounded by $\frac{10T}{\lambda} + 10$. We can then combine the finding from the two figures to conclude that $(\frac{10T}{\lambda} + 10) * 2^{\frac{\lg N}{\lg 10000}} = (\frac{5T}{\lambda} + 5) * 2^{\lg N - 3}$.

V. Related work

Releasing private data to become available at a future point in time is a challening problem. It has drawn the attention of researchers for more than two decades. Most existing works are based on cryptographical techniques. The most common approach among them is the Timed-Release Encryption (TRE). It was first proposed by May [11] in 1993. The traditional TRE is based on either time-lock puzzle scheme [14] or interactive trusted third party [11], [14]. Although the later one has been updated to a non-interactive curious third party [12], [2] and more efficient models and extensions based on that have been proposed [4], [6], [9], the central third party still makes the system suffer from a central point of trust. Besides TRE, some efforts try to model the time by capturing 'real-world-time' [8], [16]. In [8], the role of time server can be eliminated by emulating the real-world time in a computational model based on Block-chain.

Another set of research efforts related to our work is the self destructing data systems represented by the Vanish system [7]. Vanish sends data to a DHT network, Vuze. Since Vuze automatically deletes the stored data every 8 hours, the data can be self destructed. However, because of Sybil attack and hopping attack, Vanish is not safe, which was shown in [22] and later improved in [23]. Vanish employs the use of DHT to delete data, which takes advantage of the decentralization and large scale features of DHT to provide natural defenses for various attacks. In our work, we address the inverse problem of the problem addressed in Vanish, which takes advantage of those feature of DHT to keep data secure before the release time and send the data automatically after the release time. Our preliminary work on this topic [3] has studied the use of onion routing [13] and conventional whole-package replication [1] to support self-emergence of data in a DHT. While the approach presented in [3] provides the basic attack resilience to release ahead and drop attacks, it results in a significantly low churn resilience. In this paper, we develop the novel use of nested erasure coding to ensure a higher level of the churn resilience of the system. As we demonstrated in the experimental results, the proposed schemes significantly improve the resilience of the proposed protocol to churn in DHTs for a wide range of emerging times.

VI. Conclusion

In this paper, we propose *Emerge*, a self-emerging timed data release protocol for supporting self-emerging data using large-scale Distributed Hash Table (DHT) networks. The proposed schemes allow the data sender to securely hide the encryption keys of the encrypted private data stored in clouds such that the encrypted data becomes available for decryption at the defined release time but remains unavailable prior to the release time. We present a suite of routing path construction schemes for securely storing and routing encryption keys in DHT networks that protect an adversary from inferring the encryption key prior to the release time (*release-ahead attack*) or from destroying the key altogether (*drop attack*). Our experimental evaluation using Overlay Weaver DHT emulator toolkit demonstrates that the proposed schemes are resilient to both *release-ahead attack* and *drop attack* as well as to attacks that arise due to traditional churn issues in DHT networks.

References

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

[2] 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.

[3] Chao L, Balaji P. Timed-release of Self-emerging Data using Distributed Hash Tables. Proc. of 37th IEEE International Conference on Distributed Computing Systems (ICDCS), 2017.

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

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

[9] 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, 88-108, 2011.

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

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

[12] 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.

[13] 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.

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

[15] Rodrigues, Rodrigo, and Barbara Liskov. High availability in DHTs: Erasure coding vs. replication. International Workshop on Peer-to-Peer Systems, 2005.

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

[17] Shudo, Kazuyuki, Yoshio Tanaka, and Satoshi Sekiguchi. Overlay weaver: An overlay construction toolkit. Computer Communications, 31.2: 402-412, 2008.

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

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

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

[21] Weatherspoon, Hakim, and John D. Kubiatowicz. Erasure coding vs. replication: A quantitative comparison. International Workshop on Peer-to-Peer Systems, Springer Berlin Heidelberg, 2002.

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

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