Self-emerging Data Infrastructures

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

Chao Li
Beijing Key Laboratory of Security and
Privacy in Intelligent Transportation
Beijing Jiaotong University
Beijing, China
Email: li.chao@bjtu.edu.cn

***Abstract*—Timed release of data is an important security primitive for various applications. Such timed data release can be supported using a self-emerging data infrastructure that protects the data until a prescribed data release time and automatically releases to the recipient at the release time. While straight-forward centralized approaches such as cloud storage services may provide a straight-forward solution to implement self-emerging data release, unfortunately, they are limited to a single point of trust and involves a single point of control. In this vision paper, we discuss and review new decentralized designs of self-emerging data release systems using large-scale peer-to-peer (P2P) networks as the underlying infrastructure. We analyze the design of decentralized self-emerging data release systems using two P2P network infrastructures, namely Distributed Hash Tables (DHTs) and blockchains. We demonstrate how self-emerging data release can be used to support gradual release of private data in a decentralized infrastructure. Finally, we present some promising directions of future research on security primitives and protocols for timed release of private data in decentralized environments.**

I. Introduction

Timed release of data is an important security primitive for various applications. Timed data release can be supported using a self-emerging data infrastructure that protects the data until a prescribed data release time and automatically releases to the recipient at the release time. The hidden private data appears automatically to the data recipient at the release time without any assistance from the data sender. Such data are referred to as self-emerging data [30], [31], [32], [33]. Self-emerging data is widely found in practice. Examples include secure auction systems (bidding information needs protection until all bids arrive), copyrights-aware data publishing (data is automatically released when the copyright expires), secure voting mechanisms (votes are not allowed to be accessed until the end of the polling process), and post/tweet scheduler (content is scheduled to automatically post at optimal times). In the above examples, self-emerging data is released in an all-or-nothing manner, indicating that the complete information carried by the hidden data is revealed at a single release time. Self-emerging data may also get gradually released through multiple release times, allowing the carried information to be gradually revealed over time. Examples include data of individuals with privacy requirements that relax over time [28]. For instance, personal data of individuals (e.g., location trajectory patterns, shopping patterns, travel history) collected during their lifetime may be sensitive during the childhood and youth life of an individual, however, the sensitivity of such data may decrease as the individual ages and may drop significantly after the end of the individual's life and a few decades after the end of the individual's life.

Centralized storage systems such as cloud storage services [1], [3], [4] may provide a simple and straight-forward approach for implementing self-emerging data release. The storage service provider may simply keep the sensitive data until the prescribed release time and make it available at the release time. However, such a centralized approach significantly limits the data protection to a single point of trust and a single point of control. Even in cases when the service providers are trustworthy, such centralized models may lead to channels of attacks beyond the control of service providers for an adversary to breach the security and privacy of the data. It includes insider attacks [7], [41], external attacks on the centralized data infrastructure, malware and large-scale denial-of-service attacks [2], [5].

In this vision paper, we discuss and review new decentralized designs of self-emerging data release systems using large-scale peer-to-peer (P2P) networks as the underlying infrastructure. We analyze the design of decentralized self-emerging data release systems using two P2P network infrastructures, namely Distributed Hash Tables (DHTs) and blockchains. We believe that the properties of large-scale peer-to-peer (P2P) networks help resolve the challenges encountered by the previous methods. The key idea is to securely hide the shares of the key used to encrypt the data in a Distributed Hash Table (DHT) or a blockchain network and enable the key shares to automatically appear at the predetermined release time so that the protected encrypted private data can be decrypted at the release time. Highly distributed solutions for self-emerging data using large-scale P2P networks such as Distributed Hash Tables (DHTs) networks prevent the adversary from obtaining the shares of the encryption key dispersed in the P2P network before the legitimate release time. The routing mechanisms route the encryption key on the infrastructure in a deterministically pseudorandom 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, this approach using large-scale P2P networks significantly increases the attack resilience offered by the scheme.

Blockchain technologies provide additional support for decentralized implementation of self-emerging data through the use of smart contracts (e.g. Ethereum blockchain networks). Recent designs have developed a credible and enforceable smart contract for supporting self-emerging data release. The smart contract employs a set of Ethereum peers to jointly follow the proposed timed-release service protocol by allowing the participating peers to earn the remuneration paid by the service users. Through a careful design of the smart contract based on game theory, often the best choice of any rational Ethereum peer in such techniques is to always honestly follow the correct protocol.

Unlike traditional data infrastructures, the notion of timed data release in self-emerging data infrastructures allows to support dynamic utility of self-emerging data in which data users can access finer information from the published data as the data becomes older and as data privacy requirements change over time. We demonstrate this feature of self-emerging data infrastructures through a suite of dynamic data utility techniques for self-emerging data based on data privacy models. While data encryption techniques provide all or nothing protection on the data, dynamic data utility techniques ensure a more gradual release of information as the data ages and as privacy requirements change.

The rest of the paper is organized as follows. In Section II, we discuss the background and preliminaries about self-emerging data and gradual data release. In Section III, we discuss the state-of-the-art techniques for supporting self-emerging data using Distributed Hash Table (DHT) networks. Section IV discusses techniques for self-emerging data using smart contracts in peer-to-peer blockchain networks. We demonstrate the application of self-emerging data for gradual data release in Section V. In Section VI, we discuss related work. We present future research directions in Section VII and we conclude in Section VIII.

II. Background and Preliminaries

In this section, we discuss the key ideas behind self-emerging data and present the challenges of building a self-emerging data infrastructure to support timed-release of private data. We also introduce the concepts related to supporting dynamic utility of self emerging data based on data privacy models.

A. Self-emerging Data Infrastructure

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. As discussed earlier, 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. Specifically, the adversary can obtain the key prior to the release time which can violate the intended privacy provided by self-emerging data. Recent designs of self-emerging data systems employ a highly distributed solution using Distributed Hash Table (DHT) networks [18], [50] and Blockchains [42], [54] that prevent the adversary from obtaining the shares of the encryption key from the self-emerging data infrastructure before the legitimate release time. The choice of DHTs and Blockchains as the underlying storage system is motivated by the facts that DHTs and Blockchains are huge-scale geographically distributed systems that make complete decentralization possible and they are inherently designed to be reliable and robust to failures. Here, the encryption key is split into multiple fragments through Shamir's secret share scheme [49] and erasure coding [53] and routed along a pre-determined pseudorandom path from the sender to the receiver so that the key can be recovered exactly at the release time by the receiver.

A self-emerging data infrastructure consists of four major entities, namely the data sender, the data receiver, the data infrastructure 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, the data sender wants to send a message out at start time t_s and requires it to be accessible by the data receiver only after the release time t_r, where $t_s < t_r$. In a self-emerging data release system, the data sender, Alice, encrypts her message with a secret key and then sends the secret key to the self-emerging data infrastructure and the encrypted message to the cloud respectively at start time. After t_s, there is no further involvement required from Alice.
- **Data receiver**: The data receiver is an entity that wants to get access to the message sent by the data sender. In a self-emerging data release 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 self-emerging data infrastructure 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 self-emerging data infrastructure before t_r, which makes him an adversary in that case.
- **Self-emerging data infrastructure network**: A distributed self-emerging data infrastructure 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. A distributed self-emerging data infrastructure network can be constructed out of a peer-to-peer distributed hash table network (DHT) or a peer-to-peer blockchain network. This distributed network may be either a public network such as a public DHT or a public blockchain (for public self-emerging data infrastructures) or a private blockchain [6] that operates within the Intranet of an enterprise (for private self-emerging data infrastructures). To hold the secret key, during T,

,

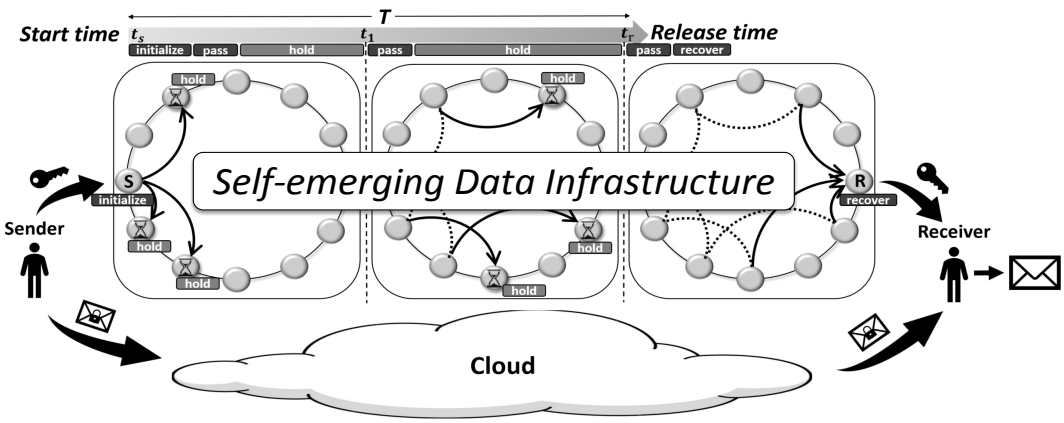

Fig. 1: Self-emerging Data Infrastructure

the self-emerging data infrastructure network should protect and maintain the existence of the secret key and prevent the loss of it. In order to hide the secret key within the self-emerging data infrastructure, the peer-to-peer network should prevent any possible attacks on the secret key from being extracted by any entities before time, t_r. In some cases, the adversary may even include Bob, who is the actual recipient. To ensure sufficient attack resilience, the self-emerging data infrastructure may split the secret key into n key shares through Shamir secret share scheme [49] and route them along carefully determined routing paths such that the key shares are transmitted from the source to the intended recipient with high attack resilience, preventing an adversary from accessing it before the release time, t_r.

- **Cloud**: A cloud storage is required to store the encrypted data during T. The encrypted data is always accessible by the authenticated data receivers.

We note that a self-emerging data infrastructure needs to demonstrate a strong defensive performance towards powerful adversaries controlling a fraction of the nodes in the distributed data infrastructure network. An adversary may try to forcibly extract the key before the actual release time. Such adversaries can also sabotage the 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*). The release-ahead attack aims to furtively extract the secret key from the self-emerging data infrastructure 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. In a 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. To launch an effective drop attack, the adversary may drop the secret key shares from reaching the intended recipients at the release time by controlling a significant number of nodes in the distributed peer-to-peer data infrastructure network. A successful drop attack will make the encrypted data inaccessible even after the release time t_r.

B. Supporting Dynamic Utility of Self-emerging Data

A self-emerging data infrastructure can provide support for dynamic utility of self-emerging data. Unlike data encryption techniques that only support *all or nothing* protection on the data, dynamic data utility techniques can ensure a more gradual release of information as the data ages in the underlying self-emerging data infrastructure.

- **Dynamic Data Utility based on Gradual Data release:** One approach to supporting dynamic utility of self-emerging data is to limit the data release based on data privacy models [13], [14] and gradually increase the disclosure as privacy requirements change. Such time-varying data utility would allow data analysts and data users to access finer and richer information on the self-emerging data as the data gets older. For instance, in a self-emerging data infrastructure, the self-emerging data may provide a lower utility at time t_1 based on a stricter ϵ_1- *differential privacy* [13], [14] and a higher utility at time t_2 based on a relaxed ϵ_2- *differential privacy* where $t_1 < t_2$ and $\epsilon_1 < \epsilon_2$. To effectively support such dynamic data utility, a self-emerging data infrastructure should be able to securely store and facilitate the timed-release of information while guaranteeing the confidentiality, integrity and availability of the data until the release time.
- **Dynamic Utility using Group and Multi-level Data Disclosure:** Self-emerging data can also support dynamic utility based on group and multi-level data disclosure. A good example of group data disclosure can be explained using a dataset describing the set of drugs purchased by patients living in different zipcodes. Group information disclosure in general may refer to some information about a group of individuals in a dataset such as the total number of 'psychiatric' drugs purchased by patients living in zipcode 15206. Dynamic data utility based on group and multi-level disclosure allows data users to access the data at different access levels as the data ages. For example, in a dataset describing the set of drugs purchased by patients in different zipcodes of the country, initially the data users may be allowed to obtain group information for only larger

groups of the population (e.g., the number of purchases of 'psychiatric' drugs in the city of 'Pittsburgh'). As the data gets older, the self-emerging data infrastructure may release access to smaller population groups (e.g., the number of 'psychiatric' drug purchases in the zipcode '15206' within 'Pittsburgh'). Finally, when the data sufficiently ages, the self-emerging data infrastructure may release access to even individual's information (e.g., how many 'psychiatric' drugs were purchased made by 'Bob' ?).

The focus of our research along these dynamic data utility techniques is to develop new methods and schemes that can support such gradual release of information in a self-emerging data infrastructure.

III. Self-emerging Data using DHT Networks

In this section, we present and discuss the state of the art techniques for supporting self-emerging data using Distributed Hash Table (DHT) networks. We begin by introducing the key concepts behind DHT networks.

A. Distributed Hash Table (DHT) Networks

DHT protocols enable efficient node lookup and message transfer services for large-scale P2P networks. In classical structured DHT overlay protocols such as Chord [50], each node maintains links to a few neighbors (maximum $O(\log\ n)$ neighbors in a network of size n). Messages are routed and retrieved as (key,value) pairs. Specifically, a message corresponding to a given key can be routed from any node in the network to a node owning the ID hash closest to the key within $O(\log\ n)$ hops. DHT also allows nodes in the P2P network to efficiently communicate and transfer data with other nodes. A DHT node can be considered as a machine with abilities of storage and communication and with the freedom to join and leave the network at any time. In addition, a DHT node has the ability to communicate with other nodes through private channels established with basic cryptography techniques and also the ability to locally store data received from other nodes.

B. Timed Data Release using DHT Networks

By leveraging the features of DHT, we have recently developed DHT-based systems [30], [31] for supporting self-emerging data that ensure that private data provided as input by a DHT node (i.e., sender) is securely hidden and inaccessible until the release time. It enables the self-emergence of the private data to a target DHT node (i.e., recipient) at the release time. More concretely, the system allows private data be packaged, split into multiple fragments through erasure coding [53] and secretly routed among the DHT nodes along a pre-determined pseudo-random routing path pattern to the recipient so that the data can be recovered exactly at the release time by the recipient.

Adversary models: We note that the security of the system 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 [51] in DHTs constitute significant challenges to the design of the system. Thus, ensuring high resilience to churn and to release-ahead attacks and drop attacks is a central objective of the system design. In [30], DHT nodes are divided into two categories, namely honest nodes and malicious nodes. In short, honest nodes may join and leave the network at any time but they will follow the designed protocols as expected when they are in the network. In addition, data locally stored at honest nodes are unavailable to the adversary. In contrast, malicious nodes are the ones controlled by the adversary and these nodes can arbitrarily violate the designed protocols and data locally stored at these nodes are known by the adversary.

At the core of the technical solutions for building the expected system over DHT is the use of redundancy. DHT nodes go in and out of the underlying P2P network frequently and there is no general and effective approach to enforce or incentivize behaviours performed by DHT nodes. As a result, requesting a single DHT node to store the hidden data is not a robust way of hiding private data in the DHT network because the data will get lost once that DHT node leaves the network. Therefore, to make data survive in a highly dynamic DHT network, data needs to be replicated to make the replicas get stored at different nodes so that the original data can still be available at the release time even if a fraction of the storage nodes become unavailable. By jointly considering challenges of churn and attacks, the protocols designed in [30] leverage erasure coding [53] to split private data into a group of fragments and allows the fragments to keep moving through the DHT nodes without sticking to fixed positions.

The protocols consist 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 selects 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 secret key to the recipient at the release time t_r.

Based on the adopted routing path construction schemes, three different protocols were proposed in [30] and all of them are based on the erasure coding mechanism [53]. As a common mechanism to protect data, erasure coding [53] can divide a data package into m fragments and re-code them into n fragments so that the package can be recovered from any m fragments ($m \leq n$).

- **One-hop path pattern**: This baseline protocol applies erasure coding to establish multiple one-hope paths between sender and recipient to guarantee attack resilience.

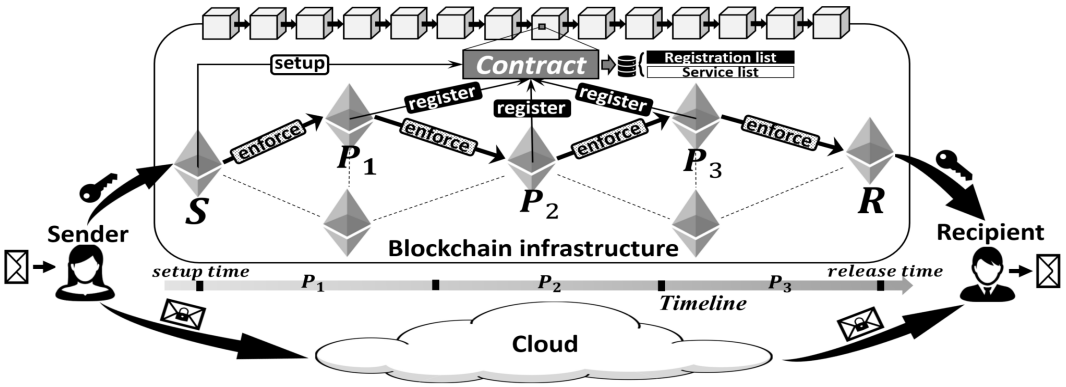

Fig. 2: Blockchain-based decentralized self-emerging data release system

- **Adjusted one-hop path pattern**: By taking dead nodes (churn) into account, this scheme tries to make the system resilient to churn issues by estimating the number of dead nodes and adjusting the parameters of erasure coding based on the estimation.
- **Multi-hop path pattern**: By dividing the entire emerging time period $T = t_s - t_r$ into several shorter time periods, this scheme iteratively implements the erasure coding mechanism to route the fragments of secret key so that the loss of fragments during each shorter time period can be suppressed and the lost fragments can be recovered by multiple usages of erasure coding.

The performance of these three schemes were compared in [30] by measuring the release-ahead attack resilience, R_r as the probability that an adversary fails to restore the secret key before the legitimate release time t_r, and drop attack resilience, R_d as the probability that an adversary fails to prevent the secret key from being restored by the recipient at the release time t_r. To evaluate the attack resilience, a DHT node is marked as malicious with probability p. To evaluate the churn resilience, each DHT node was set with a lifetime that follows an exponential distribution. The experimental results demonstrate that the 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. More concretely, all the three schemes work well for short emerging time period T. The adjusted one-hop scheme maintains good performance for medium T, but only the multi-hop scheme works well for long and very long T. However, due to the inherent design of DHTs, we consider that it may be hard to enforce the peers to honestly follow the protocol without any violations. Therefore, the DHT-based system proposed in [30], [31] mainly relies on the redundancy and recovery mechanism to make the system resilient to attacks and churn, which could result in low efficiency and high storage and communication cost. In the next section, we analyze another design of decentralized self-emerging data release systems, which leverages the decentralized consensus and monetary incentive/deposit offered by blockchain to enhance protocol enforceability.

IV. Self-emerging Data using Blockchain Smart Contracts

In this section, we discuss the state-of-the-art techniques for self-emerging data using smart contracts in peer-to-peer blockchain networks. We begin by introducing blockchains.

A. Blockchains

A blockchain is a distributed ledger maintained by a P2P network, which publicly records data as a chain of data blocks with each block containing the hash of its previous block. To falsify blockchain data, adversaries must hold enormous resources (e.g., computation power in Proof-of-Work consensus protocol [42], amount of stake in Proof-of-Stake consensus protocol [26]) to compete with the resources owned by the rest of network. The difficulty to successfully launch such attacks ensures that blockchain data are verifiable and permanent and thus provides decentralized trust among nodes in P2P networks.

B. Timed Data Release using Blockchains

Recent research on self-emerging data [32], [33] employs a blockchain infrastructure [54] that offers more robust and attractive features including decentralized democratized trust and higher fault tolerance. Similar to the DHT-based approach, the blockchain-based system also makes private data be protected until a prescribed data release time and be automatically released to the legitimate recipient at the release time, even if the data sender goes offline. In a blockchain network, such as Ethereum, a blockchain node may imply two types of nodes, namely an External Owned Account (EOA) controlled by a user through a private/public key pair or a Contract Account (CA) controlled by a smart contract. Due to the different properties associated with the two types of recipients, such as the abilities of actively participating in the protocol or locally storing data, the underlying protocols designed for the self-emerging data release system should also be different. Therefore, the blockchain-based system investigates the scenario that both the data sender and recipient are EOA accounts managed by users. In the rest of this section, for ease of presentation, we denote the data sender

and recipient as S and R while the rest of EOA accounts as P that represent peers in the Ethereum network.

There are also major differences between assumptions made in DHT-based and blockchain-based systems in terms of adversary models. Instead of following the assumption made in Section III that the peers (i.e., DHT nodes or EOA accounts) are either honest or malicious, the unique feature of blockchains produced by the blending of smart contracts and cryptocurrency makes it possible to assume that all the peers are adversaries with rationality[12], [19], [20], [43]. In DHT, due to the lack of trust among nodes, there is no possibility that one node can enforce another node to do anything. However, in Ethereum blockchain, since the decentralized trust has been established through blockchain, each smart contract can be considered as a (virtual) trusted third party. Then, the protocols in the blockchain infrastructure could be programmed as enforceable smart contracts [54]. Through careful design of the smart contract, it is possible to make rational participants follow incentives by penalizing violators through confiscation of their security deposits locked in smart contracts. More concretely, by asking each participant of a smart contract to deposit a certain amount of cryptocurrency as the security deposit to the contract, any fraudulent or dishonest behavior that violates the agreements recorded in the contract will make the violator be monetarily penalized, which incentivizes all rational participants to honestly follow the contract.

The smart contract implementation in [33] recruits a set of Ethereum peers to jointly follow the proposed self-emerging data release service protocol allowing the participating peers to earn the remuneration paid by the service users. The recruited peers need to pay security deposits so that any detected misbehaviors can result in the deposits being confiscated. The problem was modeled as an extensive-form game with imperfect information to protect against *post-facto attacks* including *drop attack* and *release-ahead attack*. Through careful design of the smart contract based on game theory, it was demonstrated that the best choice of any rational Ethereum peer is to always honestly follow the correct protocol. Specifically, the protocol proposed in [33] consists of four key components:

- **Peer registration**: At any point in time, a new peer P can register by paying a security deposit to join a contract by adding into the *registration list* maintained by the contract. This process makes the entire network learn that the peer has registered and can provide services during its prescribed working times. For example, in Figure 2, we find that P_1, P_2 and P_3 have been registered before the setup time t_s.
- **Service setup**: At any point in time, a sender S can pay remunerations and submit peers selected from the registration list to a contract C and set up a self-emerging data release service. This process makes the service to be recorded by a *service list* maintained by the contract, C. In Figure 2, we find that sender S requests a service at t_s with selected peers P_1, P_2 and P_3.
- **Service enforcement**: After a service has been set up, the participants, namely sender S, recipient R and peers, Ps should follow the protocol honestly in order to render the service successfully. Behaviors violating the protocol will lead to service failure and such misbehaviors are detected and penalized by the contract. In Figure 2, the process of routing the encrypted secret key from S to R through the path formed by the three peers is enforced by the contract C through paying remunerations for honest behaviors while confiscating deposits for misbehaviors detected by C.
- **Reporting mechanism**: To effectively detect misbehaviors in the protocol implemented in the smart contract, the reporting mechanism incentives peers to report misbehaviors by announcing an award in the contract.

The efficacy and attack-resilience of the proposed techniques were validated in [33] through rigorous analysis and experimental evaluation on the Ethereum official test network. The experiments demonstrate the low monetary cost and the low time overhead associated with the proposed approach and validate its guaranteed security properties.

While our discussions in Sections III and IV have primarily focused on design techniques for supporting self-emerging data using DHT and blockchain networks, in the next section, we discuss and demonstrate an important application of self-emerging data, namely supporting dynamic data utility using gradual data release.

V. Gradual Release of Self-emerging Data

Unlike traditional data infrastructures, the notion of timed data release in self-emerging data infrastructures allows to support dynamic utility of self-emerging data in which data users can access finer information from the published data as the data becomes older and as data privacy requirements change over time. While data encryption techniques provide all or nothing protection on the data, dynamic data utility techniques ensure a more gradual release of information as the data ages and as privacy requirements change.

In general, there are three ways to provide private data input to the self-emerging data release system:

Plaintext: As the basic option, the sender of private data can directly input the plaintext of private data to the system. Since the data needs to be stored by the P2P nodes and also routed among the nodes, such a straightforward solution may result in high cost of storing and transferring large-size private data, which in turn affects both security and scalability of the self-emerging data release system.

Encryption key: To eliminate the drawbacks of using plaintext, one option is to encrypt the private data with a key and only input the encryption key to the system. Such an encryption-based scheme can be adopted in both all-or-nothing release and gradual release models. Specifically, in the all-or-nothing release, private data only needs to be released once. Therefore only a single snapshot of encrypted data needs to be maintained by either data sender or recipient in order to make the data available at the time when the encryption key is released. In Figure 3, we show a framework of using *encryption keys* for gradual release that was proposed in a recent work [35]. In this framework, at the initial time

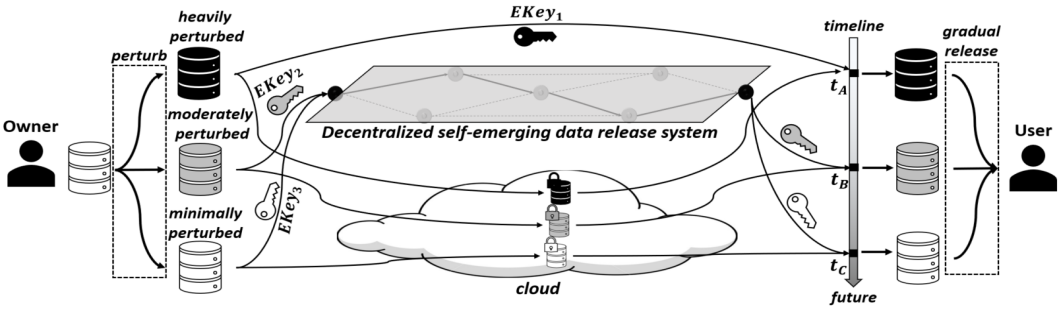

Fig. 3: Using *encryption keys* for gradual release of private data

point t_A, the data owner (i.e., sender) can use a specific privacy-preserving data perturbation technique (e.g., [8], [9], [10], [13], [15], [23], [25], [36], [21]) over the private data for multiple times so that multiple snapshots of the private data with different perturbation levels (and thus different utility levels) can be generated. After encrypting all the snapshots with different encryption keys (denoted as $EKey$), the encryption keys (except $EKey_1$) and encrypted snapshots should be sent into the decentralized self-emerging data release system and a cloud storage platform respectively. Specifically, $EKey_1$ can be directly used by the data user (i.e., recipient) to get the most heavily perturbed snapshot. After that, the data privacy level may decrease as time goes on. At a future time point t_B, the encryption key $EKey_2$ may be released by the decentralized self-emerging data release system, allowing the data user to decrypt the moderately perturbed snapshot in cloud and obtain more useful information from it. Similarly, at an even later time point t_C, the released encryption key $EKey_3$ will allow the user to gain further information from the minimally perturbed snapshot.

Perturbation key: The key idea behind the perturbation key approach is to employ a new class of reversible perturbation techniques that can use *perturbation keys* (denoted as $PKey$) to pseudo-randomly perturb data so that these keys, once released in future, can be used to directly de-perturb the single snapshot held by the user to reduce its perturbation level. The study in [35] found that a major limitation of using *encryption keys* is the cost for storing multiple encrypted snapshots of the dataset. For example, if Amazon S3 cloud storage service is used (0.023 USD/GB per month) to release a 100 GB snapshot each month for one year (i.e. release the first snapshot in the first month and store the remaining eleven snapshots and in the second month, release the second snapshot and store the remaining ten snapshots etc.), the storage cost will be close to 150 USD. Such a high storage cost is unnecessary and instead an alternate cost-effective approach can be employed as shown in Figure 4. The perturbation level of the snapshots is reversible here as it can be reduced by *perturbation keys* in future[35] . In Figure 4, at t_A, with the reversible perturbation techniques, the *perturbation key* $PKey_2$ pseudo-randomly perturbs the minimally perturbed snapshot into the moderately perturbed snapshot. Then, $PKey_1$ further pseudo-randomly perturbs the moderately perturbed snapshot to create the heavily perturbed snapshot. At this phase, all the snapshots except the most heavily perturbed one can be deleted and only the encrypted heavily perturbed snapshot needs to be stored in the cloud. Also, the data owner needs to send the *perturbation keys* into the decentralized self-emerging data release system while sending the encryption key of the heavily perturbed snapshot directly to the data user. After that, at future time point t_B, with the released $PKey_1$, the user can de-perturb the heavily perturbed snapshot to obtain the moderately perturbed snapshot. Similarly, at t_C, the minimally perturbed snapshot can be recovered from the moderately perturbed snapshot using the released $PKey_2$. Compared with the approach of using *encryption keys*, this technique needs to maintain only one snapshot as the 'seed' of all other snapshots and thus significantly reduces the cost.

Recently, there have been several efforts on implementing a cost-effective approach of using *perturbation keys* to support gradual release of data. This problem has been studied in the context of two kinds of data namely association data [35], [44] and location data [29], [34]. At the core of these techniques is the use of pseudo-randomness created by *perturbation keys* in the privacy-preserving data perturbation mechanisms. In order to make the process of data perturbation reversible, *perturbation keys* could be applied as the seeds of a generator of pseudo-randomness and replace any randomness involved in conventional data perturbation mechanisms with such pseudo-randomness so that the same keys, when released by the self-emerging data release system, could be used by the recipients to reverse the perturbation process and reduce the perturbation level.

Association data: Private association data can be perturbed and published via privacy-preserving data disclosure schemes, where the raw data is perturbed to meet the privacy requirements. However, conventional privacy-preserving data disclosure schemes focus on publishing a single snapshot of a dataset that offers a fixed privacy level that fails to directly support data release in cases when data users have different levels of access on the published dataset [13], [15], [23], [25]. In order to cost-effectively leverage gradual release of association data, techniques for multi-level reversible association data perturbation have been proposed in [35], [44], [45]. These techniques use *perturbation keys* to control the sequential generation of multiple snapshots of the perturbed data to offer multi-level access based on privacy levels. It requires only the *perturbation keys* to be sent into the

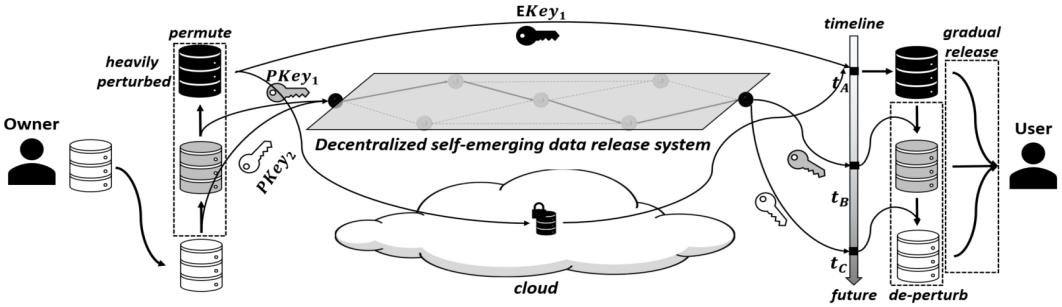

Fig. 4: Cost-effective gradual release of private data by using *perturbation keys*

self-emerging data release system while only a single snapshot of perturbed dataset needs to be maintained.

Location data: Reversible perturbation techniques for location data can also be used for supporting gradual data release. In general, location anonymization refers to the process of perturbing the exact location of users as a spatially cloaked region such that a user's location becomes indistinguishable from the location of a set of other users. Recent techniques called *ReverseCloak* [29], [34] provide a new class of reversible spatial cloaking mechanisms that effectively support multi-level location privacy protection. These techniques allow de-anonymization of the cloaking region when corresponding *perturbation keys* are released to the data users in the future through the self-emerging data release system. Such techniques are shown to be highly efficient and scalable even while supporting the reversibility feature.

VI. Related Work

Releasing private data to the future is a challenging research topic that has intrigued researchers for more than two decades. Timed-Release Encryption (TRE) was first proposed by May [39] in 1993. The TRE schemes can be mainly classified into two groups, namely time-lock puzzle scheme [16], [47] and third party scheme [39], [47]. In 2003, a non-interactive TRE scheme[40] has been proposed based on quadratic residues (QR-TRE). Here the time server used in the approach requires to be trusted. The main drawback with these approaches is that they require a trusted centralized server (a single point of trust). Since then, even though more efficient models and extensions were developed [11], [24], [27], most of them require the time server to be trusted which becomes a security bottleneck to the overall system. Furthermore, requiring to solve a time-lock puzzle for each timed data release is not only computationally expensive but also such an approach is not scalable to a large-scale data infrastructure. In contrast, decentralized self-emerging data infrastructure does not involve a central point of trust and is highly scalable, involving only a modest computational and data transfer cost compared to these cryptographic solutions.

The problem of timed release of data using blockchain as a reference time clock has attracted the attention of the cryptographers [22], [37]. In Bitcoin [42], the difficulty of PoW is diversely adjusted to make average generation time of each block to be ten minutes, which makes blockchain to be a reference time clock with correctness guaranteed by the entire distributed network. By combining witness encryption [17] and blockchain [22], [37], one can leverage the computation power of PoW in the blockchain to decrypt a message after a certain number of new blocks have been generated. While this is an interesting idea based on cryptography, current implementations of witness encryption are far from practical, requiring an astronomical decryption time estimated to be 2^{100} seconds [37]. In contrast to these cryptographic solutions, the key idea behind decentralized self-emerging data infrastructures is to leverage the huge scalability, proof-of-work (PoW) and distributed ledger features of P2P blockchain networks to develop a scalable self-emerging data release protocol. By combining the attack-resilience and provable correctness and stability of the self-emerging data release protocols with blockchains, decentralized infrastructures provide a highly practical distributed system guaranteeing security, scalability and computational cost-efficiency for timed data release.

Current data privacy solutions are limited to the static notion of privacy and do not consider the time-varying aspect of privacy requirements supported through gradual release of self-emerging data. While both differential privacy and k-anonymity models have been extensively studied for sensitive data publication [13], [14], [48], [52], [38], recent work has developed solutions to support multi-level privacy [29], [46], [34] and group privacy [45] with static privacy requirements. The work closest to the dynamic data utility techniques supported by self-emerging data is [28] which proposes an optimized differentially private data release for releasing information after users relax their privacy requirements. These techniques primarily aim to improve query accuracy in an interactive data release setting when a privacy requirement is relaxed. Moreover, they are centralized approaches involving a single point of trust. In contrast, a self-emerging data that allows a gradual release of information supports dynamic data utility using a timed data release model.

VII. Future Research Directions

Future research on self-emerging data infrastructures may develop techniques for enforcing dynamic controls on the protected data, including techniques for changing access rights and access privileges in real-time. For instance, the original data may be released into the data infrastructure to be released

at a certain point of time and at a later point in time, the data owner may decide to change the time of release. A dynamic access control will allow such changes in the self-emerging data release protocol. Similarly, a data owner may want to revoke access to a self-emerging data after it is published in the data infrastructure. These features will require new techniques for real-time access revocation and methods to support real-time modifications of the data release time in the self-emerging data infrastructure. Such techniques will enable the infrastructure to speed up and slow down the data emergence based on dynamically changing preferences. Similarly, an additional dynamic control feature would be to provide an ability to release self-emerging data based on external event triggers. Such an event driven mechanism will allow protected data to be released during an emergency or critical event based on the event-based access rights set by the data owner. Another direction of future research may address an important challenge in blockchain-based self-emerging data infrastructure solutions namely to minimize the gas cost. In Ethereum, any transaction that creates new smart contracts or calls functions of existing smart contracts will spend gas that costs real money. A possible approach to tackle this problem is to differentiate old service providers from new service providers by establishing a trust management mechanism that leverages the service history of each provider to compute a trust score. The number of involved service providers can then be dynamically adjusted based on their trust scores in order to reduce the number of selected service providers when most of them maintain high trust scores.

VIII. Conclusion

In this vision paper, we reviewed and analyzed new decentralized designs of self-emerging data release systems using two kinds of large-scale peer-to-peer (P2P) networks namely Distributed Hash Tables (DHTs) and blockchains. The timed data release system design using DHTs leverages the efficient lookup service of DHT for securely storing and routing the self-emerging data in the DHT network before a prescribed data release time. However, due to the lack of ways to enforce the required protocol behaviors by the peers in the DHT, the DHT-based system usually needs complex routing paths involving a large number of nodes to offer sufficient data redundancy. To resolve this issue, the timed data release system using blockchains leverages the decentralized trust and the native cryptocurrency offered by blockchains to enforce that peers honestly follow their agreements through cryptocurrency-driven monetary incentive and penalty. With the assumption that all peers are rational, the protocols designed based on game theory can make rational nodes choose to honestly comply with the protocols. We also reviewed and analyzed new mechanisms for supporting cost-effective gradual release of self-emerging data in a decentralized infrastructure to support dynamic utility of data. While data encryption techniques provide all or nothing protection on the data, dynamic data utility techniques ensure a more gradual release of information as the data ages and as privacy requirements change.

Reference

[1] Amazon simple storage service (s3):. https://aws.amazon.com/s3/.

[2] Aws best practices for ddos resiliency:. https://d0.awsstatic.com/whitepapers/DDoS_White_Paper_June2015.pdf.

[3] Google cloud storage:. https://cloud.google.com/storage/.

[4] Microsoft azure storage:. https://azure.microsoft.com/en-us/services/storage/.

[5] Semantec report: The continued rise of ddos attacks:. http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/the-continued-rise-of-ddos-attacks.pdf.

[6] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, et al. Hyperledger fabric: a distributed operating system for permissioned blockchains. In *Proceedings of the Thirteenth EuroSys Conference*, page 30. ACM, 2018.

[7] CERT Insider Threat Center. Us state of cybercrime survey (2014), 2014.

[8] Rui Chen, Gergely Acs, and Claude Castelluccia. Differentially private sequential data publication via variable-length n-grams. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 638–649. ACM, 2012.

[9] Rui Chen, Benjamin Fung, Bipin C Desai, and Nériah M Sossou. Differentially private transit data publication: a case study on the montreal transportation system. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 213–221. ACM, 2012.

[10] Rui Chen, Noman Mohammed, Benjamin CM Fung, Bipin C Desai, and Li Xiong. Publishing set-valued data via differential privacy. *Proceedings of the VLDB Endowment*, 4(11):1087–1098, 2011.

[11] Jung Hee Cheon, Nicholas Hopper, Yongdae Kim, and Ivan Osipkov. Provably secure timed-release public key encryption. *ACM Transactions on Information and System Security (TISSEC)*, 11(2):4, 2008.

[12] Changyu Dong, Yilei Wang, Amjad Aldweesh, Patrick McCorry, and Aad van Moorsel. Betrayal, distrust, and rationality: Smart counter-collusion contracts for verifiable cloud computing. *ACM CCS*, 2017.

[13] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *TCC*, volume 3876, pages 265–284. Springer, 2006.

[14] Cynthia Dwork, Aaron Roth, et al. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3–4):211–407, 2014.

[15] Arik Friedman and Assaf Schuster. Data mining with differential privacy. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 493–502. ACM, 2010.

[16] Juan A Garay and Markus Jakobsson. Timed release of standard digital signatures. In *International Conference on Financial Cryptography*, pages 168–182. Springer, 2002.

[17] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 467–476. ACM, 2013.

[18] Roxana Geambasu, Tadayoshi Kohno, Amit A Levy, and Henry M Levy. Vanish: Increasing data privacy with self-destructing data. In *USENIX Security Symposium*, volume 9, 2009.

[19] Adam Groce and Jonathan Katz. Fair computation with rational players. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 81–98. Springer, 2012.

[20] Siyao Guo, Pavel Hubáček, Alon Rosen, and Margarita Vald. Rational sumchecks. In *Theory of Cryptography Conference*, pages 319–351. Springer, 2016.

[21] Xi He, Graham Cormode, Ashwin Machanavajjhala, Cecilia M Procopiuc, and Divesh Srivastava. Dpt: differentially private trajectory synthesis using hierarchical reference systems. *Proceedings of the VLDB Endowment*, 8(11):1154–1165, 2015.

[22] Tibor Jager. How to build time-lock encryption. *IACR Cryptology ePrint Archive*, 2015:478, 2015.

[23] Vishesh Karwa, Sofya Raskhodnikova, Adam Smith, and Grigory Yaroslavtsev. Private analysis of graph structure. *Proceedings of the VLDB Endowment*, 4(11):1146–1157, 2011.

[24] Kohei Kasamatsu, Takahiro Matsuda, Keita Emura, Nuttapong Attrapadung, Goichiro Hanaoka, and Hideki Imai. Time-specific encryption from forward-secure encryption. In *International Conference on Security and Cryptography for Networks*, pages 184–204. Springer, 2012.

[25] Shiva Prasad Kasiviswanathan, Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. Analyzing graphs with node differential privacy. In *Theory of Cryptography*, pages 457–476. Springer, 2013.

[26] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Annual International Cryptology Conference*, pages 357–388. Springer, 2017.

[27] Ryo Kikuchi, Atsushi Fujioka, Yoshiaki Okamoto, and Taiichi Saito. Strong security notions for timed-release public-key encryption revisited. In *International Conference on Information Security and Cryptology*, pages 88–108. Springer, 2011.

[28] Fragkiskos Koufogiannis, Shuo Han, and George J Pappas. Gradually releasing private data under differential privacy. 2015.

[29] Chao Li and Balaji Palanisamy. Reversecloak: Protecting multi-level location privacy over road networks. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pages 673–682. ACM, 2015.

[30] Chao Li and Balaji Palanisamy. Emerge: Self-emerging data release using cloud data storage. In *Cloud Computing (CLOUD), 2017 IEEE 10th International Conference on*, pages 26–33. IEEE, 2017.

[31] Chao Li and Balaji Palanisamy. Timed-release of self-emerging data using distributed hash tables. In *Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on*, pages 2344–2351. IEEE, 2017.

[32] Chao Li and Balaji Palanisamy. Decentralized privacy-preserving timed execution in blockchain-based smart contract platforms. In *2018 IEEE 25th International Conference on High Performance Computing (HiPC)*, pages 265–274. IEEE, 2018.

[33] Chao Li and Balaji Palanisamy. Decentralized release of self-emerging data using smart contracts. In *2018 IEEE 37th Symposium on Reliable Distributed Systems (SRDS)*, pages 213–220. IEEE, 2018.

[34] Chao Li and Balaji Palanisamy. Reversible spatio-temporal perturbation for protecting location privacy. *Computer Communications*, 135:16–27, 2019.

[35] Chao Li, Balaji Palanisamy, and Prashant Krishnamurthy. Reversible data perturbation techniques for multi-level privacy-preserving data publication. In *International Conference on Big Data*, pages 26–42. Springer, 2018.

[36] Ninghui Li, Tiancheng Li, and Suresh Venkatasubramanian. t-closeness: Privacy beyond k-anonymity and l-diversity. In *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, pages 106–115. IEEE, 2007.

[37] Jia Liu, Flavio Garcia, and Mark Ryan. Time-release protocol from bitcoin and witness encryption for sat. *IACR Cryptology ePrint Archive*, 2015:482, 2015.

[38] Ashwin Machanavajjhala, Johannes Gehrke, Daniel Kifer, and Muthuramakrishnan Venkitasubramaniam. l-diversity: Privacy beyond k-anonymity. In *Data Engineering, 2006. ICDE'06. Proceedings of the 22nd International Conference on*, pages 24–24. IEEE, 2006.

[39] Timothy May. Timed-release crypto. *http://www. hks. net. cpunks/cpunks-0/1560. html*, 1992.

[40] Marco Casassa Mont, Keith Harrison, and Martin Sadler. The hp time vault service: Innovating the way confidential information is disclosed, at the right time. 2002.

[41] Andrew P Moore, Dawn M Cappelli, and Randall F Trzeciak. The "big picture" of insider it sabotage across us critical infrastructures. In *Insider Attack and Cyber Security*, pages 17–52. Springer, 2008.

[42] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[43] Thanh Hong Nguyen, Rong Yang, Amos Azaria, Sarit Kraus, and Milind Tambe. Analyzing the effectiveness of adversary modeling in security games. In *AAAI*, 2013.

[44] Balaji Palanisamy, Chao Li, and Prashant Krishnamurthy. Group differential privacy-preserving disclosure of multi-level association graphs. In *Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on*, pages 2587–2588. IEEE, 2017.

[45] Balaji Palanisamy, Chao Li, and Prashant Krishnamurthy. Group privacy-aware disclosure of association graph data. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 1043–1052. IEEE, 2017.

[46] Balaji Palanisamy and Ling Liu. Privacy-preserving data publishing in the cloud: A multi-level utility controlled approach. In *2015 IEEE 8th International Conference on Cloud Computing*, pages 130–137. IEEE, 2015.

[47] Ronald L Rivest, Adi Shamir, and David A Wagner. Time-lock puzzles and timed-release crypto. 1996.

[48] Pierangela Samarati. Protecting respondents identities in microdata release. *IEEE transactions on Knowledge and Data Engineering*, 13(6):1010–1027, 2001.

[49] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[50] Ion Stoica, Robert Morris, David Karger, M Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM Computer Communication Review*, 31(4):149–160, 2001.

[51] Daniel Stutzbach and Reza Rejaie. Understanding churn in peer-to-peer networks. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, pages 189–202. ACM, 2006.

[52] Latanya Sweeney. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(05):557–570, 2002.

[53] Hakim Weatherspoon and John D Kubiatowicz. Erasure coding vs. replication: A quantitative comparison. In *International Workshop on Peer-to-Peer Systems*, pages 328–337. Springer, 2002.

[54] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper*, 151:1–32, 2014.