Keridemlia: A Discovery Mechanism for KERI

Ryan W. West
ryan@ryanwwest.com

Samuel M. Smith Ph.D.
sam@prosapien.com

Abstract

KERI (Key Event Receipt Infrastructure) is a fully decentralized, end-to-end verifiable identity system that uses identifiers containing public keys to prove ownership of messages sent by them. Among other properties, each identifier is bound to a separately-stored key event log (KEL). Key rotation and revocation can be performed by appending events to this log, and always-available, self-run witness servers store these events. Users that wish to verify who controls an identifier must retrieve the KEL to get the most recent public key state from this log, which requires knowing the witness's IP address.

Keridemlia is a service similar to DNS that stores key state for identifiers, which maps identifiers to witness identifiers and witness identifiers to witness IP addresses. It uses a modified distributed hash table, Kademlia, to store this data across many nodes for horizontal scalability and high performance. Keridemlia provides an API for publishing and retrieving each type of mapping. It only allows signed, verified data to be published to the network, discarding all unverified data.

Background

The Internet today is inherently insecure. It started out as an experimental communication network between universities and governments in 1969 [1], taking decades to evolve into the globally interconnected web we rely on today. During its lengthy development, few anticipated the scale to which the Internet would grow or how critical it would become, and as such, security and privacy were not prioritized.

This began to change in the mid-1990s, as Internet usage began to pick around that time and cyberattacks were becoming more common. Security researchers began to focus on ways to protect users of the existing system. In 1995, they released the Hypertext Transfer Protocol Secure (HTTPS), which secures the connection between a client and a website's server, encrypting traffic between the two endpoints. HTTPS relies on Certificate Authorities (CAs) to issue secure X.509 certificates [8] to every website which they use to validate their identity and ownership of the site. Two years later, they released Domain Name System Security Extensions (DNSSEC), a solution meant to secure DNS, the system that maps domain names (such as www.example.com) to their respective IP addresses (such as 93.184.216.34).

These systems and many others have been introduced and refined for decades since, but hackers and criminals continue to find and exploit vulnerabilities at an alarming rate, causing hundreds of billions of dollars of estimated damages per year with customer data breaches,

fraud, and identity theft [2]. Each vulnerability is unique, but they all share a common weakness—they rely on centralized, bolt-on solutions meant to secure a system that was originally designed without security in mind.

KERI (Key Event Receipt Infrastructure) is an end-to-end verifiable identity system in active development that, unlike other security solutions, recasts the system design with security first and foremost at its core [5]. Unlike HTTPS, which must rely on the integrity of Certificate Authority companies to sign certificates, KERI is completely decentralized in nature and instead relies purely on cryptographic operations. To do this, KERI utilizes Autonomic Identifiers (AIDs)—special, unique identifiers that are derived directly from the public-private keypair(s) that an owner, known as the *controller*, can sign or encrypt data with, as well as prove ownership over the AID. This means every AID is *self-certifying*—anyone can verify the legitimacy and ownership of an AID with only the AID and a corresponding log of key events associated with the AID, without the need of a CA-issued certificate [3]. AIDs are also *self-managing*, meaning they have the ability to rotate the public-private keypair associated with them as needed over time [7].

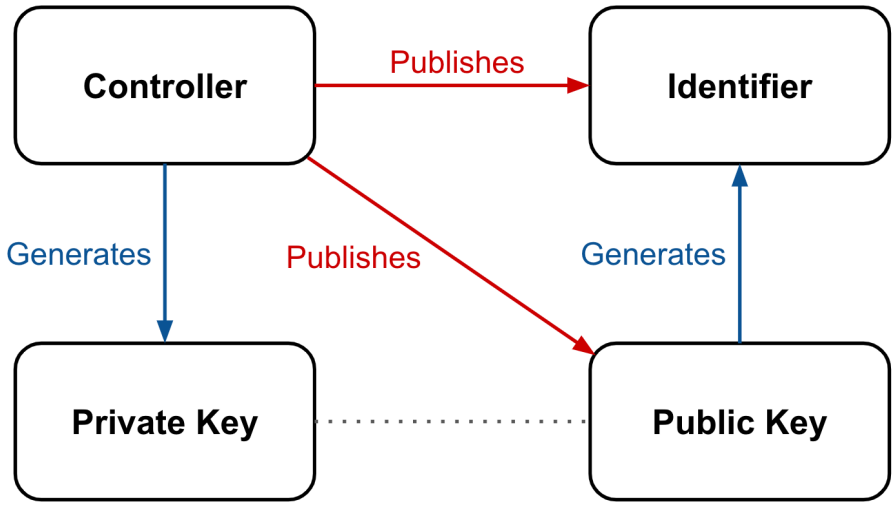

Figure 1: Anatomy of an Autonomic, Self-Certifying Identifier (AID) [3]

KERI also provides a key management infrastructure to use with AIDs which allows for self-governed, automatic key rotations, delegations, and revocations, run entirely on user-owned hardware if desired. It does not rely on any external system, blockchain, ledger, or company—KERI gives the user complete control of their online identity and property. Despite this, KERI can still quickly detect duplicitous behavior, such as if a user or their servers are dishonest.

KERI can be run in two modes: direct and indirect. In direct or private mode, two users communicate directly using each other's pre-exchanged IP addresses. In indirect or public mode, identity owners, known as controllers, must know the mappings of other controllers' AIDs

to their corresponding IP addresses in order to securely communicate and verify control. If KERI were to become a viable alternative to the existing Internet infrastructure, most controllers would utilize indirect mode for their interactions. This presents a need for a discovery mechanism that can map a controllers' AID to an IP address.

Keridemlia: Discovery for KERI

Keridemlia is a distributed system that stores mappings of AIDs to IP addresses, enabling controllers to use KERI in indirect mode. Keridemlia leverages a distributed hash table (DHT) to map AIDs to witness AIDs and witness AIDs to witness IP addresses [4]. In indirect mode, a *witness* is an always-running service that keeps a log of a controller's key event history (such as key rotation or delegation). Witnesses can be run on the controller's own infrastructure, on a centralized infrastructure, or even on a blockchain, making KERI as centralized or decentralized as each controller prefers. Verifying who controls an identifier or similarly controlled digital property in the namespace controlled by that identifier, requires contacting one of the identity's witnesses via its IP address. Keridemlia acts similar to a DNS server in that it can map both names (or AIDs) to other names (like a CNAME record) and names to IP addresses (like an A record). KERI is interoperable, meaning its identifiers are not confined to one system but could be used across many. Because KERI identifiers themselves contain all needed information or pointers to information to verify who controls them, Keridemlia may serve as an end-verifiable replacement for DNS.

Design

At a high level, Keridemlia runs an API wrapper on top of a Python Kademlia implementation. Kademlia is a popular DHT that evenly distributes storage of key-value pairs across all participating nodes in the network. Because nodes can freely join or leave the network, it can be configured to redundantly cache copies of the pairs on multiple nodes to lower the chance of data loss. This additionally provides horizontal scalability, as increasing data storage requirements can be met by increasing node count with little additional overhead. For Keridemlia, the key-value pairs consist of controller AID -> witness AID mappings (but this actually maps to additional data which will be explained later) and witness AID -> witness IP mappings. Keridemlia is interfaced such that Kademlia could be easily switched out with another DHT like mechanism or library, if desired.

Data Structure Research

Keridemlia is designed to be highly performant, distributed, and scalable. Databases are traditionally stored on single servers, but are quickly overwhelmed by both high read/write traffic and limited storage capacity. Experts in virtually all computing fields respond by distributing the traffic and storage across many servers, which term is known as horizontal scalability. However, there are many ways to build a distributed system. We researched several types and implementations of these systems before settling on Kademlia. This involved considering 3 primary ways to build a distributed system:

1. **Hierarchy**. A hierarchical distributed system uses several levels of servers to provide scalable service. Servers at the bottom level accept requests from users and forward requests to higher layers that may each contain parts of the data required for the response. However, servers also *cache* or temporarily store the responses from higher-layer servers, and if a server receives a request that it has cached, rather than using bandwidth to contact the next server, it can just return the cached response. These systems are typically centralized, meaning all servers are controlled by one entity. DNS is based on a hierarchy of servers; for the domain cs.byu.edu, there exist separate DNS servers to cache results for “cs”, “byu”, and “edu”.

Hierarchical systems are highly performant and scale well. However, because they are centralized and rely on few ‘root’ servers at the top of the hierarchy (such as DNS’s authoritative name servers), they are prone to network attacks and system outages if only a few servers are taken down.

2. **Flooding**. Servers using flooding algorithms, also known as broadcast algorithms, send packets to every server they are connected to in hopes of eventually reaching a destination server. Each server connected to the first then forwards the same packet to all of their neighbors (excluding the source server). This pattern continues until a hop count associated with the packet reaches 0. The *hop count* represents the number of ‘hops’ a packet should make between servers until stopping and should be high enough to reach the destination server. The packet in question essentially ‘floods’ the network even after the packet may have reached the destination server.

Flooding algorithms were popular in the 2000’s and used by Peer-to-peer (P2P) file-sharing systems such as Napster, as they are simple to implement and can lead to quickly locating data. They can also be easily decentralized as many networks today allow anyone to spin up their own server and join the network. However, they do not scale well on their own. Increasing the number of nodes connected to the network polynomially increases the bandwidth used every time a packet is sent, which can become highly inefficient and slow.

From these inefficiencies stemmed more optimized systems which used gossip algorithms to decrease throughput. Nodes running gossip algorithms only send packets to some of their peers, and the choice is often random. While this significantly improves scalability, this sort of optimized flooding still tends to perform worse than other distributed systems at scale and may increase request latency due to the random nature of gossiping.

3. **Distributed Hash Tables (DHT)**. A hash table is a data structure that maps keys to values. Hash tables provide constant-time lookup by running the provided key through a hash function that outputs a fixed-length digest. This digest points to the location of the value. Some common hash functions used are MD5, SHA-1, and SHA-256. A *distributed* hash table expands the storage of key-value pairs to multiple nodes and uses routing algorithms to store and retrieve data from specific nodes depending on the hash created. As the number of nodes increase, the average time taken to find the destination node only logarithmically increases. DHTs can be configured to store several redundant copies of data and can also easily rebalance data when nodes are added or removed from the service pool. DHTs’ redundancy and decentralization

make it difficult to attack; with redundancy, taking down a few nodes is very unlikely to lead to data loss and other nodes can quickly be added to maintain network performance.

Another consideration was using blockchain-based systems such as the Sovrin identity-based ledger. However, despite claiming to be decentralized, these consortium-based blockchains still operate under a number of companies and are not fully decentralized in nature. In addition, Sovrin and other blockchains also face numerous scalability and performance issues that have not yet been fully solved. Current identity-based blockchain systems also force users to use their system, contrary to the interoperable nature of KERI. Blockchains provide an immutable data store that relies on the presence of all past events. However, the European Union's General Data Protection Regulation (GDPR) "Right to be Erased" requires companies to delete personally identifiable information (PII) if the owner requests it, which presents a problem since the blockchain is immutable (unchangeable) [9]. Rather than facing these problems, we opted for a distributed system that can be run entirely on user-owned hardware, requiring zero external dependencies if the owner prefers.

After evaluating these types of distributed system, we decided to use a DHT because of its performance, scalability, decentralization, and reliability. The flooding systems' lack of scalability made it a poor choice. DHTs' routing algorithms are not as simple to implement as hierarchy-based systems, thus requiring more work for them to achieve the same performance levels, but their reliability and decentralization nonetheless made them a better choice.

There are many DHT designs available that vary in how they come to consensus between nodes, how traffic is routed, how data is stored, and in several other ways. Of these options, we chose to use Kademlia, a popular DHT designed and tested in 2002 which has seen widespread usage and proven performance over time. It has an open-source, simple, actively developed Python implementation available on Github which suitably matches the current KERI Python library [2].

Step 1: Integrating KERI with Kademlia

KERI's Python library implements a special database class which keeps track of all events, keys, and other metadata needed to use KERI. This class is backed by Lightning Memory-Mapped Database (LMDB), a high-performance key-value storage [6]. Because LMDB and Kademlia are both key-value databases and the KERI storage class can keep track of all necessary information, I decided to modify Kademlia's backend to use KERI's storage class. This class also provides additional methods for verification that are helpful at later points in development.

Step 2: Building an API

To understand the needs of the API, some background is needed of how KERI allows key rotations. A *key rotation* replaces an existing public-private keypair with a new keypair and is routinely done to improve security or is manually carried out after keys have been compromised. KERI allows for key rotation for all *transferable* AIDs by utilizing their immutable, signed key event logs (KELs) which are cryptographically bound to each AID and keep track of all history

for the AID (see Figure 2). The *inception* event is always the first event and defines both the current public key and lists the AIDs of all witness servers. This event also records a *nxt* or “next” key which contains a cryptographic commitment to a future public key held by the controller. This acts as a *prerotation* to the log and allows the controller to append a new *rotation* event to the log which both replaces the current keys and adds a new “next” key commitment for future rotations.

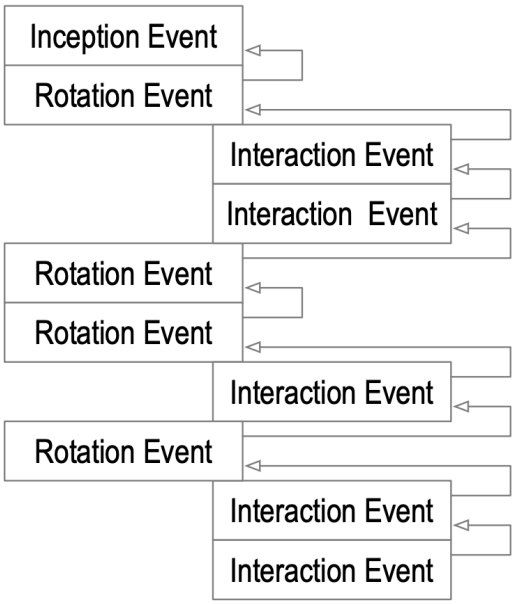

Figure 2: Sample Key Event Log (KEL)

There are other types of events that can be appended to the KEL, called *interaction* events, which can also change the witness servers or delegate authority to other identifiers. All events must be signed. Replaying each event in the KEL in chronological order with the KERI Python library produces a *key event state* or snapshot of the log which is a key-value pair map containing all necessary information to verify the AID at the present time. This replay also verifies that the signature of each event is correct at the time of publishing to prevent an attacker from appending a duplicitous event. [11] gives an example of a transferable prefix KES and other KERI structures.

Unlike a traditional Kademlia DHT, which allows any key-value pairs to be published to the network, Keridemlia currently only stores information necessary to map controller’s Autonomic Identifiers (AIDs) to their KES, which contain the current set of witness AIDs, and each of those witness AIDs to IP addresses. Keridemlia could map controller AIDs to their entire KEL, but these take up much more space than the resulting KES that they produce, so a KES is more practical. To make this idea more clear and simplify life for developers, I designed a simple application programming interface (API) which exposes four HTTP endpoints that can be called. These endpoints are found in *Table 1* below.

HTTP Method	Endpoint	Description
POST	/id (requires json request body)	publish controller AID -> key event state (KES) mapping
GET	/id/<aid>	get KES from controller AID
POST	/ip (requires json request body)	publish witness AID -> witness IP mapping
GET	/ip/<witness_id>	get witness IP from witness AID

Table 1: Keridemlia Endpoints

There are two sets of endpoints: one related to AID -> KES mappings, and one related to non-transferable AID -> IP address mappings. Each set has one endpoint to publish a new mapping and one endpoint to get an existing value given a key.

Some AIDs, such as witness AIDs, are *non-transferable*, meaning their inception event's *nxt* or "next" key is blank and will not allow for any additional rotations or other events. In the event that these AIDs are compromised, they are simply abandoned and replaced. A major benefit of non-transferable AIDs is that they can be verified purely by having the AID, as it innately contains the public key necessary to verify the only keys it is associated with.

Witnesses exist partially because the controller needs a place to publish the current version of the KEL associated with their transferable AID. A transferable AID by itself cannot be verified if the key has been rotated, revoked, or otherwise modified. Both determining this and getting new information for verification requires fetching a fresh copy of the KEL from the AID's witness.

At this point, Keridemlia becomes necessary. When a controller wants others to be able to use their AID, they first publish the AID's key event state using the "/id" post endpoint. This contains all associated witness AIDs which are each published to the DHT. Additionally, each witness must publish mappings of their non-transferable AID to a signed copy of their current IP address using the "ip" post endpoint. With both mappings in place, a consumer can use each get endpoint to pair every published AID with its current witnesses' IP addresses in order to verify the AID.

One could argue that because Keridemlia contains each controller AID's KES, there is no point in actually contacting the witness thereafter to get the KEL, since the KEL itself produces the KES snapshot that is already available. However, this presents two issues. First, while the controller should keep their AID -> KES mapping up-to-date, it may not be, which requires checking the KEL from a witness to be sure. Second, a malicious Keridemlia node could publish false KES data to mislead clients. Replaying the KEL also verifies the signatures of each event and allows the caller to detect any dishonesty.

Step 3: Verifying data is authentic

KERI is based on the concept or property of end-verifiability—all events can be verified anytime by anyone. Keridemlia supports the property of end-verifiability in its implementation. Keridemlia should avoid storing data for retrieval in the first place unless it can be verified as authentic using its public key. Because the current key event state (KES) is provided for transferable AIDs, Keridemlia can verify AIDs by using the public key present in their state. The KERI Python library provides several classes and functions which can use this information to verify each identifier. Non-transferable AIDs for witnesses, in contrast, cannot be rotated, meaning the public key embedded in the identifier can be used to verify the identifier. These do not need additional data for verification. IP addresses must also have a signature from the witness to ensure that only witnesses can add mappings of their AID to their current IP address.

This verification does more than save storage space by disallowing garbage data—it also provides a number of security guarantees not present in other DNS or DHT systems. Because all data is signed by the appropriate keys, it prevents cache poisoning on an honest server. Cache poisoning occurs when intentionally fake data is placed on a server in order to deny service to users or even mislead them with wrong data. This has been a common pitfall of DNS since it by default allows new mappings to be published without verification.

After data has been verified Keridemlia stores the signatures. Every subsequent HTTP GET request provides those signatures, allowing clients to independently verify the results. This prevents a client from ever using an invalid entry if it were somehow published to the DHT. It also limits attacks by malicious Keridemlia servers to only dropping data entries or denying service.

Potential Optimizations

The Keridemlia prototype implements the first two design steps presented above and functions as a simple replacement for a DNS server. It does not completely implement the third step of verifying the authenticity of all data before publishing it to the ledger, but this is being actively worked on. However, there are many improvements that can increase its throughput capacity, scalability, and usability. Some of these improvements are listed below.

If a controller updates the witnesses that their AID points to by adding an event to the KEL, this updates the latest key event state and makes the corresponding AID -> IP mappings stored in the DHT out of date. The controller and their witnesses may need to republish their mappings

whenever this happens. This will likely be done automatically by the witness servers but will need to be implemented.

Keridemlia will most commonly be used to get an witness IP address for a given AID. This currently requires two HTTP requests: one to get a witness AID, and another that uses that AID to get its IP address. Keridemlia could be modified to perform both lookups for one request, lowering the bandwidth requirements and decreasing the time to get the IP address.

Keridemlia may also benefit from local caching of data. The modified Kademlia DHT distributes all data between every server in the network, which means much of the requests require several hops across the network to get the right data. Similar to a DNS server, a local cache could store the mappings of the latest requests. Keridemlia could then check for the data in this local cache before making a Kademlia network request.

In addition to cryptographic AIDs, KERI supports Legitimized Human Meaningful Identifiers (LIDs) which are more easily recognized and remembered by people. In the future, Keridemlia could be changed to support mappings of AIDs to corresponding LIDs, and could be further opened to support more key-value pairs as they become necessary.

Related Work

There are other efforts to decentralize DNS, such as the experimental Handshake network [10]. Handshake employs a decentralized naming protocol, using a blockchain to provide a globally unique namespace that anyone can use. The blockchain effectively decentralizes control of the namespace and uses proof of work as a cost function to disincentivize domain name squatting and spam. However, unlike Keridemlia, it only provides one scarce namespace, requiring users to compete for available names. Additionally, it requires the user to rely on third-party hardware and the integrity of the network for availability and privacy, which may be undesirable.

Conclusion

By accounting for key rotation, revocation, and other security from the ground up, KERI provides an excellent, scalable alternative to today's identity solutions. Its complete decentralization places all control in cryptographic mathematics, rather than having to rely on the trust of other companies and entities. Keridemlia makes KERI usable in indirect mode by providing a key-value store that ultimately maps autonomic identifiers to their respective witness server IP addresses. This system also acts as a secure, decentralized replacement for DNS systems. By using Keridemlia with KERI, this creates a scalable system that reinvents the security and privacy of the internet.

References

1. Camille Paloque-Bergès & Valérie Schafer (2019) Arpanet (1969–2019), Internet Histories, 3:1, 1-14, DOI: 10.1080/24701475.2018.1560921

2. The White House, The Cost of Malicious Cyber Activity to the U.S. Economy. 2018. https://www.whitehouse.gov/wp-content/uploads/2018/03/The-Cost-of-Malicious-Cyber-Activity-to-the-U.S.-Economy.pdf

3. Alex Preukschat & Drummond Reed. Self-Sovereign Identity: Decentralized Digital Identity and Verifiable Credentials. Chapter 10: Decentralized Key Management. Unpublished Manuscript.

4. Samuel M. Smith. Key Event Receipt Infrastructure (KERI) Design. Work in progress. https://github.com/SmithSamuelM/Papers/blob/master/whitepapers/KERI_WP_2.x.web.pdf.

5. Maymounkov, Petar, & David Mazieres. "Kademlia: A peer-to-peer information system based on the xor metric." International Workshop on Peer-to-Peer Systems. Springer, Berlin, Heidelberg, 2002.

6. http://www.lmdb.tech/

7. Samuel M. Smith. Universal Identifier Theory. Work in Progress. https://github.com/SmithSamuelM/Papers/blob/master/whitepapers/IdentifierTheory_web.pdf

8. Cooper, D., Santesson, S., Farrell, S. et al., “RFC 5280 Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile,” IETF, 2008/05/01 https://tools.ietf.org/html/rfc5280

9. GDPR Right to erasure (‘right to be forgotten’): https://gdpr-info.eu/art-17-gdpr/

10. The Handshake Protocol Wiki. https://hsd-dev.org/guides/protocol.html

11. KID0003 Subsection Compact Labels https://github.com/decentralized-identity/keri/blob/master/kids/kid0003_compact_labels.md#transferable--prefix-signer-key-state-message