Encrypted Data Vaults

a white paper from Rebooting the Web of Trust IX

by Amy Guy, David Lamers, Tobias Looker, Manu Sporny, and Dmitri Zagidulin
with Daniel Bluhm and Kim Hamilton Duffy

We store a significant amount of sensitive data online, such as personally identifying information (PII), trade secrets, family pictures, and customer information. The data that we store is often not protected in an appropriate manner.

Legislation, such as the General Data Protection Regulation (GDPR), incentivizes service providers to better preserve individuals' privacy, primarily through making the providers liable in the event of a data breach. This liability pressure has revealed a technological gap, whereby providers are often not equipped with technology that can suitably protect their customers. Encrypted Data Vaults fill this gap and provide a variety of other benefits.

This paper describes current approaches and architectures, derived requirements, design goals, and dangers that implementers should be aware of when implementing data storage. This paper also explores the base assumptions of these sorts of systems such as providing privacy-respecting mechanisms for storing, indexing, and retrieving encrypted data, as well as data portability.

CURRENT ECOSYSTEM AND EXISTING WORK

The problem of decentralized data storage has been approached from various different angles, and personal data stores (PDS), decentralized or otherwise, have a long history in commercial and academic settings [1, 2]. Different approaches have resulted in variations in terminology and architectures. The diagram below shows the types of components that are emerging, and the roles they play. Encrypted Data Vaults fulfill a *storage* role.

Sponsors for the Rebooting the Web of Trust IX Design Workshop

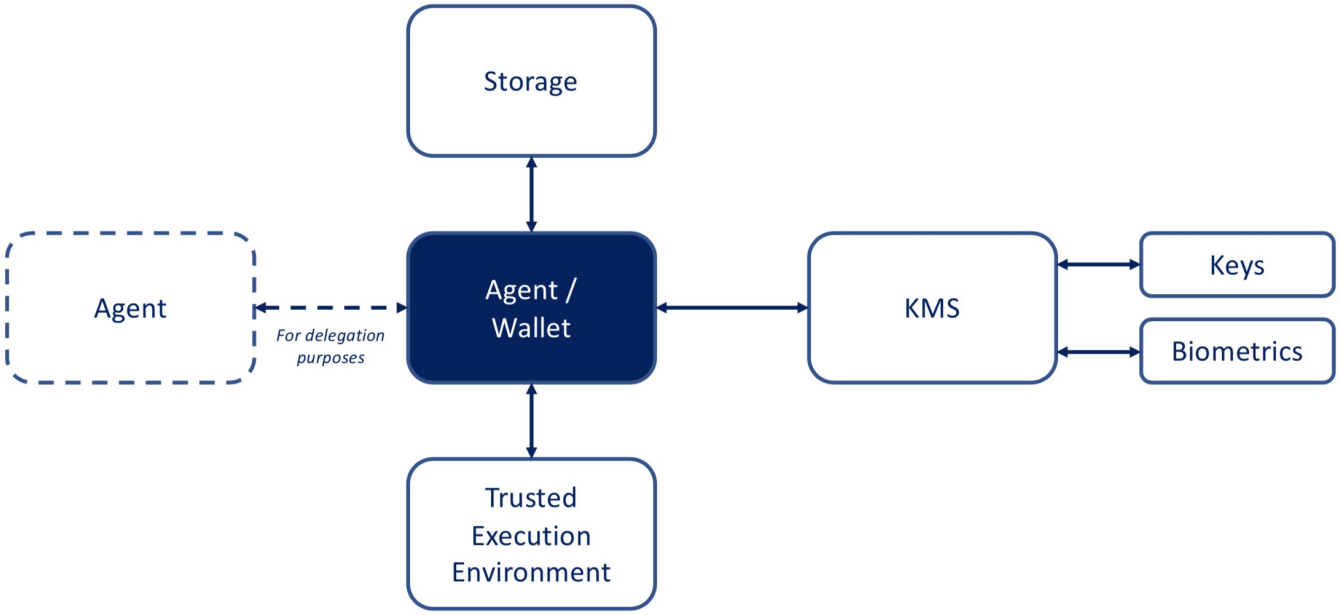

Figure 1: Roles and interactions

What follows is an outline of commonalities and differences between a selection of existing implementations. This list is by no means comprehensive, but we tried to choose projects which are representative of the different types of approaches out there, as well as, for practical reasons, choosing ones with which the authors are most familiar.

Architectures and deployment

Many architectures are designed around the idea of separating storage of data from a layer of applications which make use of the stored data. We can think of these applications as clients, with varying levels of complexity, and the data stores as servers. Some projects expect an ecosystem of diverse applications to emerge, and design their protocols with this in mind.

NextCloud, Solid, and DIF's Identity Hubs all describe architectures for decoupling end-user applications from data storage. Such applications may be generic file management interfaces for browsing or sharing data, or specialized domain specific tools designed for particular tasks (e.g., a calendar). Datashards, Tahoe-LAFS, and IPFS are only concerned with data storage and retrieval.

In the case of Solid, NextCloud, and Identity Hubs, end users have the option of installing and running the server portion of the data store on a device they control, or signing up to an already configured instance hosted by a trusted third-party (eg. a commercial provider, affiliated institution, or friend). For Datashards and Tahoe-LAFS, end users install a native application on one or more device(s) they control, and data is stored locally to these devices. IPFS is peer-to-peer, so end users only install the read/write client, and data is stored across a public network.

Identity Hubs are responsible for additional things beyond just data storage, for example management of the end user's profile; transmission of human- or machine-readable messages through the Actions interface; or pointers to external services.

Project	Type	Data organized into	Extended file metadata supported?	Encryption required?	Metadata encrypted?	Query on metadata?
DIF Identity Hubs	Document-based data store	Collections	Yes	Unclear *	No	Yes
Solid	File and graph store	Containers	Yes	No	No	No
Nextcloud	File store	Directories	Yes	No	No	Yes
IPFS	Content-addressable distributed file store	n/a	No	No	n/a	n/a
Tahoe-LAFS	Clustered distributed file system	Directories	No	Yes	n/a	n/a
Datashards	Low-level encrypted storage protocol	n/a	No	Yes	n/a	n/a

** - Unclear if Identity Hubs requires encryption. Encryption proposal paper suggests that it's optional*

Project	Read/write protocol	Authn	Access Control	Data locality	Replication
DIF Identity Hubs	Custom	DID Auth	Custom	Server	Planned
Solid	REST (LDP)	WebID-OIDC	WAC	Server	Planned
Nextcloud	WebDAV	Custom	Custom (RBAC)	Server	Yes (Enterprise version, via MySQL clustering)
IPFS	Cli / HTTP	n/a	n/a	Public nodes	Peer-to-peer
Tahoe-LAFS	Various (REST, WebDAV, others)	n/a	Capabilities	Storage cluster	Sharded chunks (multiple copies)
Datashards	cli / modular	n/a	Capabilities	Modular (server, public nodes, other)	(depends on backend)

Encryption policies

An important consideration of encrypted data stores is which components of the architecture have access to the (unencrypted) data, or who controls the private keys. There are roughly three approaches: storage-side encryption, client-side (edge) encryption, and gateway-side encryption (which is a hybrid of the previous two).

Any data storage systems that let the user store arbitrary data also support client-side encryption at the most basic level. That is, they let the user encrypt data themselves, and then store it. This doesn't mean these systems are optimized for encrypted data however. Querying and access control for encrypted data may be difficult (as is the case for Solid, NextCloud, Identity Hubs, and IPFS).

Storage-side encryption is usually implemented as whole-disk encryption or filesystem-level encryption. This is widely supported and understood, and any type of hosted cloud storage is likely to use storage-side encryption. In this scenario the private keys are managed by the service provider or controller of the storage server, which may be a different entity than the user who is storing the data. Encrypting the data while it resides on disk is a useful security measure should physical access to the storage hardware be compromised, but does not guarantee that *only* the original user who stored the data has access.

Conversely, client-side encryption - as with Datashards - offers a high level of security and privacy, especially if metadata can be encrypted as well. Encryption is done at the individual data object level, usually aided by a keychain or wallet client, so the user has direct access to the private keys. This comes at a cost, however, since the significant responsibility of key management and recovery falls squarely onto the end user. In addition, the question of key management becomes more complex when data needs to be shared.

Gateway-side encryption systems like Tahoe-LAFS take an approach that combines techniques from storage-side and client-side encryption architectures. These storage systems, typically encountered among multi-server clusters or some "encryption as a platform" cloud service providers, recognize that client-side key management may be too difficult for some users and use cases, and offer to perform encryption and decryption themselves in a way that is transparent to the client application. At the same time, they aim to minimize the number of components (storage servers) that have access to the private decryption keys. As a result, the keys usually reside on "gateway" servers, which encrypt the data before passing it to the storage servers. The encryption/decryption is transparent to the client, and the data is opaque to the storage servers, which can be modular/pluggable as a result. Gateway-side encryption provides some benefits over storage-side systems, but also share the drawbacks: the gateway sysadmin controls the keys, not the user.

Encrypted Metadata and Documents vs Blobs

> *We kill people based on metadata.*
> *- General Michael Hayden, former director of the NSA and the CIA*

Whether or not metadata can be (or is required to be) encrypted has implications for privacy, security, and usability of a system.

Some systems, including Solid, NextCloud, and Identity Hubs, support the inclusion of arbitrary metadata on binary data blobs. IPFS, Datashards, and Tahoe-LAFS do not. Solid metadata is written by clients per-resource using RDF. Identity Hubs uses JWTs for per-object metadata, as well as JSON documents for additional metadata in Collections (also the clients' responsibility). NextCloud clients can add metadata to documents using WebDAV custom properties. None of these options for including metadata allow for it to be encrypted.

Access Interface and Control

Whether data is accessed over a network or on a local device, data objects tend to need globally unique identifiers. Interfaces for reading and writing data in stores, as well as the mechanisms to restrict or grant the ability to do so, vary between implementations.

NextCloud and Solid both make use of existing Web standards. NextCloud uses WebDAV to allow client applications to read, write, and search data on the server's filesystem using a directory structure, supported by a custom login flow for authentication. Solid combines LDP with OpenID Connect authentication and Web Access Control to enable users to sign into client applications, which can then read or write data. Resources (data objects) on Solid servers are represented by HTTP URIs, and Solid servers receive HTTP requests containing RDF payloads and create or modify the target URI accordingly.

Identity Hubs uses JSON Web Tokens (JWTs) and specified endpoints. Multiple requests are required, first to retrieve metadata for the desired data object(s), and then to retrieve the sequence of commits that make up the actual data. Mechanisms for authentication are still under development. Access control is carried out via posting to the Permissions interface.

Tahoe-LAFS uses a client-gateway-storage server architecture, whereby a client passes data to a gateway server for encryption and chunking. The gateway, in turn, stores the individual chunks on a cluster of storage servers. Several copies of the data are stored, for greater availability and disaster recovery. Services are identified with Foolscap URIs, and the client can be configured to use HTTP or (S)FTP or to listen to a local directory ('magic folder') to create, update and delete data. Data is organized in a filesystem-like directory structure and access control makes use of this.

IPFS is a distributed content-addressed storage mechanism which breaks data up into Merkel-DAGs. IPFS uses IPLD to generate URIs for data from the content and to link content together on the network and uses DHTs to discover content on the network.

Indexing and Querying

Encrypted data that is opaque to the storage server introduces challenges for indexing and searching the contents of the data. Some systems work around this with a certain amount of unencrypted metadata attached to the data objects. Another possibility is unencrypted listings of pointers to filtered subsets of data.

Solid aims to provide a web-accessible interface to a file system. Resources (RDF documents or arbitrary files) are organized into folder-like Containers. Precisely how the data is stored is an implementation detail (e.g., a filesystem or a database). No search interface has been specified, but some implementations may expose a SPARQL endpoint or Triple Pattern Fragments.

Identity Hubs similarly uses a Collections interface for indexing. Clients are responsible for writing appropriate metadata to Collections, which are not themselves encrypted, enabling the Hub to respond to queries.

NextCloud sorts data objects into directories, and clients can use WebDAV `SEARCH` and `PROPFIND` to query data and metadata.

Tahoe-LAFS, Datashards and IPFS are low-level storage protocols, and do not provide for indexing or searching the data.

Availability, Replication and Conflict Resolution

Replicating data across multiple storage locations is an important resilience and security mechanism. Systems which support peer-to-peer replication must provide conflict resolution mechanisms such as CRDTs or require end-user intervention to merge files which get out of sync.

NextCloud provides spreading data across multiple instances for scalability as a commercial enterprise offering. Different NextCloud servers do not talk to each other directly, but can do so via applications installed by the user. Similarly, different instances of Solid servers do not communicate with each other; client apps can perform any communication necessary between storage servers, but these are usually servers belonging to different users, rather than stores with copies of the same data.

IPFS, Tahoe-LAFS, and Datashards achieve high availability by chunking data, using content-addressable links, and storing many copies of the chunks. They don't handle conflict resolution since they are low-level protocols and the data is opaque to the servers.

Identity Hubs synchronization of changes and conflict resolution between Hub instances is under development.

Summary

The preceding sections provided an overview of a subset of active projects in the personal data store ecosystem. These projects aim to give end users control over their data without requiring centralized authorities or

proprietary technologies.

Requiring client-side (edge) encryption for all data and metadata at the same time as enabling the user to store data on multiple devices and to share data with others, whilst also having searchable or queryable data, has been historically very difficult to implement in one system. We can see from this survey that trade-offs are often made which sacrifice privacy in favor of usability, or vice versa.

Due to a number of maturing technologies and standards, we are hopeful that such trade-offs are no longer necessary, and that it is possible to design a privacy-preserving protocol for encrypted decentralized data storage that has broad practical appeal.

CORE USE CASES

The following four use cases have been identified as representative of common usage patterns (though are by no means the only ones).

Store and Use Data

I want to store my data in a safe location. I don't want the storage provider to be able to see any data I store. This means that only I can see and use the data.

Search Data

Over time, I will store a large amount of data. I want to search the data, but don't want the service provider to know what I'm storing or searching for.

Share Data With One or More Entities

I want to share my data with other people and services. I can decide on giving other entities access to data in my storage area when I save the data for the first time or in a later stage. The storage should only give access to others when I have explicitly given consent for each item.

I want to be able to revoke the access of others at any time. When sharing data, I can include an expiration date for the access to my data by a third-party.

Store the Same Data in More Than One Place

I want to backup my data across multiple storage locations in case one fails. These locations can be hosted by different storage providers and can be accessible over different protocols. One location could be local on my phone, while another might be cloud-based. The locations should be able to synchronize between each other so data is up to date in both places regardless of how I create or update data, and this should happen automatically and without my help as much as possible.

REQUIREMENTS

This section elaborates upon a number of requirements that have been gathered from the core use cases.

Privacy and Multi-party Encryption

One of the main goals of this system is ensuring the privacy of an entity's data so that it cannot be accessed by unauthorized parties, including the storage provider.

To accomplish this, the data must be encrypted both while it is in transit (being sent over a network) and while it is at rest (on a storage system).

Since data could be shared with more than one entity, it is also necessary for the encryption mechanism to support encrypting data to multiple parties.

Sharing and Authorization

It is necessary to have a mechanism that enables authorized sharing of encrypted information among one or more entities.

The system is expected to specify one mandatory authorization scheme, but also allow other alternate authorization schemes. Examples of authorization schemes include OAuth2, Web Access Control, and Authorization Capabilities (ZCAPs).

Identifiers

The system should be identifier agnostic. In general, identifiers that are a form of URN or URL are preferred. While it is presumed that Decentralized Identifiers (DIDs) will be used by the system in a few important ways, hard-coding the implementations to DIDs would be an anti-pattern.

Versioning and Replication

It is expected that information can be backed up on a continuous basis. For this reason, it is necessary for the system to support at least one mandatory versioning strategy and one mandatory replication strategy, but also allow other alternate versioning and replication strategies.

Metadata and Searching

Large volumes of data are expected to be stored using this system, which then need to be efficiently and selectively retrieved. To that end, an encrypted search mechanism is a necessary feature of the system.

It is important for clients to be able to associate metadata with the data such that it can be searched. At the same time, since privacy of both data *and* metadata is a key requirement, the metadata must be stored in an encrypted state, and service providers must be able to perform those searches in an opaque and privacy-preserving way, without being able to see the metadata.

Protocols

Since this system can reside in a variety of operating environments, it is important that at least one protocol is mandatory, but that other protocols are also allowed by the design. Examples of protocols include HTTP, gRPC, Bluetooth, and various binary on-the-wire protocols.

DESIGN GOALS

This section elaborates upon a number of guiding principles and design goals that shape Encrypted Data Vaults.

Layered and Modular Architecture

A layered architectural approach is used to ensure that the foundation for the system is easy to implement while allowing more complex functionality to be layered on top of the lower foundations.

For example, Layer 1 might contain the mandatory features for the most basic system, Layer 2 might contain useful features for most deployments, Layer 3 might contain advanced features needed by a small subset of the ecosystem, and Layer 4 might contain extremely complex features that are needed by a very small subset of the ecosystem.

Prioritize Privacy

This system is intended to protect an entity's privacy. When exploring new features, always ask "How would this impact privacy?". New features that negatively impact privacy are expected to undergo extreme scrutiny to determine if the trade-offs are worth the new functionality.

Push Implementation Complexity to the Client

Servers in this system are expected to provide functionality strongly focused on the storage and retrieval of encrypted data. The more a server knows, the greater the risk to the privacy of the entity storing the data, and the more liability the service provider might have for hosting data. In addition, pushing complexity to the client enables service providers to provide stable server-side implementations while innovation can by carried out by clients.

ARCHITECTURE

This section describes the architecture of the Encrypted Data Vault protocol.

This document defines a client-server relationship, whereby the vault is regarded as the server and the client acts as the interface used to interact with the vault.

This architecture is layered in nature, where the foundational layer consists of an operational system with minimal features, and where more advanced features are layered on top. Implementations can choose to implement only the foundational layer, or optionally, additional layers consisting of a richer set of features for more advanced use cases.

Deployment Topologies

Based on the use cases, we consider the following deployment topologies:

- Mobile Device Only: The server and the client reside on the same device. The vault is a library providing functionality via a binary API, using local storage to provide an encrypted database.
- Mobile Device Plus Cloud Storage: A mobile device plays the role of a client, and the server is a remote cloud-based service provider that has exposed the storage via a network-based API (eg. REST over HTTPS). Data is not stored on the mobile device.
- Multiple Devices (Single User) Plus Cloud Storage: When adding more devices managed by a single user, the vault can be used to synchronize data across devices.
- Multiple Devices (Multiple Users) Plus Cloud Storage: When pairing multiple users with cloud storage, the vault can be used to synchronize data between multiple users with the help of replication and merge strategies.

Server and Client Responsibilities

The server is assumed to be of low trust, and must have no visibility into the data that it persists. However, even in this model, the server still has a set of minimum responsibilities it must adhere to.

The client is responsible for providing an interface to the server, with bindings for each relevant protocol (HTTP, RPC, or binary over-the-wire protocols), as required by the implementation.

All encryption and decryption of data is done on the client side, at the edges. The data (including metadata) MUST be opaque to the server, and the architecture is designed to prevent the server from being able to decrypt it.

Layer 1 (L1) Responsibilities

Layer 1 consists of a client-server system that is capable of encrypting data in transit and at rest.

Server: Validate Requests (L1)

When a vault client makes a request to store, query, modify, or delete data in the vault, the server validates the request. Since the actual data and metadata in any given request is encrypted, such validation is necessarily limited and largely depends on the protocol and the semantics of the request.

Server: Persist Data (L1)

The mechanism a server uses to persist data, such as storage on a local, networked, or distributed file system, is determined by the implementation. The persistence mechanism is expected to adhere to the common expectations of a data storage provider, such as reliable storage and retrieval of data.

Server: Persist Global Configuration (L1)

A vault has a global configuration that defines the following properties:

- Stream chunk size
- Other config metadata

The configuration allows the the client to perform capability discovery regarding things like authorization, protocol, and replication mechanisms that are used by the server.

Server: Enforcement of Authorization Policies (L1)

When a client makes a request to store, query, modify, or delete data in the vault, the server enforces any authorization policy that is associated with the request.

Client: Encrypted Data Chunking (L1)

An Encrypted Data Vault is capable of storing many different types of data, including large unstructured binary data. This means that storing a file as a single entry would be challenging for systems that have limits on single record sizes. For example, some databases set the maximum size for a single record to 16MB. As a result, it is necessary that large data is chunked into sizes that are easily managed by a server. It is the responsibility of the client to set the chunk size of each resource and chunk large data into manageable chunks for the server. It is the responsibility of the server to deny requests to store chunks larger that it can handle.

Each chunk is encrypted individually using authenticated encryption. Doing so protects against attacks where an attacking server replaces chunks in a large file and requires the entire file to be downloaded and decrypted by the victim before determining that the file is compromised. Encrypting each chunk with authenticated encryption ensures that a client knows that it has a valid chunk before proceeding to the next one. Note that another authorized client can still perform an attack by doing authenticated encryption on a chunk, but a server is not capable of launching the same attack.

Client: Resource Structure (L1)

The process of storing encrypted data starts with the creation of a Resource by the client, with the following structure.

Resource:

- `id` (required)
- `meta`
  ⇨ `meta.contentType` MIME type
- `content` - entire payload, or a manifest-like list of hashlinks to individual chunks

If the data is less than the chunk size, it is embedded directly into the `content`.

Otherwise, the data is sharded into chunks by the client (see next section), and each chunk is encrypted and sent to the server. In this case, `content` contains a manifest-like listing of URIs to individual chunks (integrity-protected by Hashlinks).

Client: Encrypted Resource Structure (L1)

The process of creating the Encrypted Resource. If the data was sharded into chunks, this is done after the individual chunks are written to the server.

- `id`
- `index` - encrypted index tags prepared by the client (for use with privacy-preserving querying over encrypted resources)
- *Chunk size* (if different from the default in global config)
- *Versioning metadata* - such as sequence numbers, Git-like hashes, or other mechanisms
- *Encrypted resource payload* - encoded as a `jwe`, `cwe` or other appropriate mechanism

Layer 2 (L2) Responsibilities

Layer 2 consists of a system that is capable of sharing data among multiple entities, of versioning and replication, and of performing privacy-preserving searches in an efficient manner.

Client: Encrypted Search Indexes (L2)

To enable privacy-preserving querying (where the search index is opaque to the server), the client must prepare a list of encrypted index tags (which are stored in the Encrypted Resource, alongside the encrypted data contents).

Client: Versioning and Replication (L2)

A server must support *at least one* versioning/change control mechanism. Replication is done by the client, not by the server (since the client controls the keys, knows about which other servers to replicate to, etc.). If an Encrypted Data Vault implementation aims to provide replication functionality, it MUST also pick a versioning/change control strategy (since replication necessarily involves conflict resolution). Some versioning strategies are implicit ("last write wins", eg. `rsync` or uploading a file to a file hosting service), but keep in mind that a replication strategy *always* implies that some sort of conflict resolution mechanism should be involved.

Client: Sharing With Other Entities (L2)

An individual vault's choice of authorization mechanism determines how a client shares resources with other entities (authorization capability link or similar mechanism).

Layer 3 (L3) Responsibilities

Server: Notifications (L3)

It is helpful if data storage providers are able to notify clients when changes to persisted data occurs. A server may optionally implement a mechanism by which clients can subscribe to changes in the vault.

Client: Vault-wide Integrity Protection (L3)

Vault-wide integrity protection is provided to prevent a variety of storage provider attacks where data is modified in a way that is undetectable, such as if documents are reverted to older versions or deleted. This protection requires that a global catalog of all the resource identifiers that belong to a user, along with the most recent version, is stored and kept up to date by the client. Some clients may store a copy of this catalog locally (and include integrity protection mechanism such as Hashlinks) to guard against interference or deletion by the server.

EXTENSION POINTS

Encrypted Data Vaults support a number of extension points:

- Protocol/API - One or more protocols such as library APIs, HTTPS, gRPC, or Bluetooth can be used to access the system.
- Encryption Strategies - One or more encryption strategies such as AES-GCM or XSalsa20Poly1305 can be used to encrypt data.
- Authorization Strategies - One or more authorization strategies such as OAuth2, HTTP Signatures, or Authorization Capabilities can be used to protect access to encrypted data.
- Versioning Strategies and Replication Strategies - One or more versioning and replication strategies such as counters, cryptographic hashes, or CRDTs (Conflict-free Replicated Data Types) can be used to synchronize data.
- Notification mechanisms - One or more notification mechanisms can be used to signal to clients that data has changed in the vault.

SECURITY AND PRIVACY CONSIDERATIONS

This section details the general security and privacy considerations as well as specific privacy implications of deploying Encrypted Data Vaults into production environments.

Malicious or Accidental Modification of Data

While a service provider is not able to read data in an Encrypted Data Vault, it is possible for a service provider to delete, add, or modify encrypted data. The deletion, addition, or modification of encrypted data can be prevented by keeping a global manifest of data in the data vault.

Compromised Vault

An Encrypted Data Vault can be compromised if the data controller (the entity who holds the decryption keys and appropriate authorization credentials) accidentally grants access to an attacker. For example, a victim might accidentally authorize an attacker to the entire vault or mishandle their encryption key. Once an attacker has access to the system, they may modify, remove, or change the vault's configuration.

Data Access Timing Attacks

While it is normally difficult for a server to determine the identity of an entity as well as the purpose for which that entity is accessing the Encrypted Data Vault, there is always metadata related to access patterns, rough file sizes, and other information that is leaked when an entity accesses the vault. The system has been designed to not leak information that it creates concerning privacy limitations, an approach that protects against many, but not all, surveillance strategies that may be used by servers that are not acting in the best interest of the privacy of the vault's users.

Encrypted Data on Public Networks

Assuming that all encryption schemes will eventually be broken is a safe assumption to make when protecting one's data. For this reason, it is inadvisable that servers use any sort of public storage network to store encrypted data as a storage strategy.

Unencrypted Data on Server

While this system goes to great lengths to encrypt content and metadata, there are a handful of fields that cannot be encrypted in order to ensure the server can provide the features outlined in this specification. For example, a version number associated with data provides insight into how often the data is modified. The identifiers associated with encrypted content enables a server to gain knowledge by possibly correlating identifiers across documents. Implementations are advised to minimize the amount of information that is stored in an unencrypted fashion.

Partial Matching on Encrypted Indexes

The encrypted indexes used by this system are designed to maximize privacy. As a result, there are a number of operations that are common in search systems that are not available with encrypted indexes, such as partial matching on encrypted text fields or searches over a scalar range. These features might be added in the future through the use of zero-knowledge encryption schemes.

Threat Model for Malicious Service Provider

While it is expected that most service providers are not malicious, it is also important to understand what a malicious service provider can and cannot do. The following attacks are possible given a malicious service provider:

- Correlation of entities accessing information in a vault
- Speculation about the types of files stored in a vault depending on file size and access patterns
- Addition, deletion, and modification of encrypted data
- Not enforcing authorization policy set on the encrypted data
- Exfiltrating encrypted data to an unknown external system

FUTURE WORK

The following items will be considered during our ongoing and future work on Encrypted Data Vaults:

- Query details, sorting, pagination
- Key management
- Choice of authorization strategy
- Choice of change control / conflict resolution strategy
- Notification / pub-sub mechanisms
- With respect to the authorization model, does the vault merely enforce authorization rules or act as an authorization server?
- How can end users be reassured of the trustworthiness of a vault host?
- Further analysis of potential attack vectors of malicious servers, and mitigation techniques.
- What are the opportunities for encrypted searching (Homomorphic encryption, ZKPs), and what are the dangers?
- Retrieval of the history of an object's updates

CONCLUSION

This paper has described current approaches and architectures for encrypted storage systems, provided derived requirements and design goals, and highlighted dangers that implementers should be aware of when implementing privacy-preserving data storage systems. This paper also explored the base assumptions of these sorts of systems such as providing privacy-respecting mechanisms for storing, indexing, and retrieving encrypted data, as well as data portability. The authors of this paper expect to continue to work on these concepts and contribute them to a pre-standards track specification that achieves the concepts and goals outlined in this paper.

Additional Credits

Lead Author: Amy Guy

Authors: David Lamers, Tobias Looker, Manu Sporny, and Dmitri Zagidulin

Contributors: Daniel Bluhm and Kim Hamilton Duffy

Sample APA Citation:

Guy, A., Lamers, D., Looker, T., Sporny, M. and Zagidulin, D. (2019). Encrypted Data Vaults. *Rebooting the Web of Trust IX*. Retrieved from https://github.com/WebOfTrustInfo/rwot9-prague/blob/master/final-documents/encrypted-data-vaults.pdf.

About Rebooting the Web of Trust

This paper was produced as part of the Rebooting the Web of Trust IX design workshop. On September 3rd to 6th, 2019, over 60 tech visionaries came together in Prague, The Czech Republic to talk about the future of decentralized trust on the internet with the goal of writing at least 5 white papers and specs. This is one of them.

RWOT Board of Directors: Christopher Allen, Joe Andrieu, Kim Hamilton Duffy

Members of the Organizing Committee: Dan Burnett, Dmitri Zagidulin

Sponsors: Digital Contract Design (Gold), Protocol Labs (Silver), Digital Bazaar (Ongoing Sustaining), Jolocom

Community Sponsors: Blockchain Commons, Consensys, Learning Machine, Legendary Requirements

Workshop Credits: Christopher Allen (Founder, Co-Producer), Joe Andrieu (Co-Producer and Facilitator), and Shannon Appelcline (Editor-in-chief).

Thanks to our other contributors and sponsors!

Thanks also to Paralelní Polis and the Institute for CryptoAnarchy in Prague.

What's Next?

The design workshop and this paper are just starting points for Rebooting the Web of Trust. If you have any comments, thoughts, or expansions on this paper, please post them to our GitHub issues page:

https://github.com/WebOfTrustInfo/rwot9/issues

The tenth Rebooting the Web of Trust design workshop is scheduled for early 2020. If you'd like to be involved or would like to help sponsor the event, email:

rwot-leadership@googlegroups.com