electronics

MDPI

Article

Practical Use of Secret Sharing for Enhancing Privacy in Clouds

Peter Čuřík, Roderik Ploszek and Pavol Zajac *

Department of Computer Science and Mathematics, Faculty of Electrical Engineering and Information Technology, Slovak University of Technology in Bratislava, Ilkovičova 3, 812 19 Bratislava, Slovakia
* Correspondence: pavol.zajac@stuba.sk

Abstract: The topic of this contribution is the practical use of secret sharing schemes in securing private data. In the first part, we focus on the security model and the overview of the available solutions. In the second part, we describe our solution for storing sensitive data in commercial cloud storage systems, an application called Datachest. The application uploads the data in encrypted form. Cryptographic keys are divided into shares. Each cloud receives one share. This solution improves the security of users' sensitive data in the cloud. In the final part, we explore the ramifications of secret sharing on the security and management of IoT devices, especially home appliances.

Keywords: secret sharing; privacy; cloud; IoT

1. Introduction

The Internet of Things (IoT) is rapidly increasing its presence in the lives of ordinary people. The amount of data produced by the IoT can be too large to easily manage locally, and data storage is often based on cloud infrastructure. Although this facilitates and helps IoT deployment and management, it also raises significant issues with user privacy.

A basic solution to preserve privacy is to use encryption [1]. Encryption transforms the user data into an unreadable form, and the original data can only be reconstructed with the corresponding secret key. In a cloud-based scenario, the use of encryption raises various concerns. The first is the question of key management.

The primary issue being where the decryption keys are stored. If the key is stored in the cloud, the owner of the cloud, or an attacker who gained access to the cloud, can decrypt the data on demand. If the key is stored with the user, there is a high chance of losing the key, and thus losing access to the encrypted data. The owner of the cloud can also raise a question of legitimacy, as he can be responsible in some jurisdictions for the data he stores, and which he cannot access without the key.

A solution to these types of problems can be found in cryptographic schemes under the umbrella term 'Secret sharing'. A secret sharing scheme can be used to distribute secret data into multiple shares distributed among participants. The participants can then reconstruct the original secret if they form a legitimate access structure, e.g., if at least t out of n shares are present (a so-called threshold scheme).

In our cloud situation, a secret sharing scheme can be used to distribute private data (or just a decryption key) into multiple (competing) clouds. The user can recover the data if he retains access to specific cloud providers. If the access structure is correctly specified, a security compromise of a single cloud (or a specific number of them) does not compromise the privacy of the stored data. A legitimate forensic investigation can still access the required data by collecting the shares across multiple clouds, or if some specified government service is given some master share in a specific access scheme.

Secret sharing can also be a suitable solution for GDPR concerns [2]. GDPR regulates the processing of personal data stored and processed by companies located in the EU and EEA. This also applies to companies that collect and process personal data from EU citizens outside the EU. The GDPR gives users the right to control the availability of their

Citation: Čuřík, P.; Ploszek, R.; Zajac, P. Practical Use of Secret Sharing for Enhancing Privacy in Clouds. *Electronics* **2022**, *11*, 2758. https://doi.org/10.3390/electronics11172758

Academic Editor: Cheng-Chi Lee

Received: 3 August 2022
Accepted: 30 August 2022
Published: 1 September 2022

personal data. For example, it gives them the ability to access information about how their data is processed or the ability to erase the data altogether. Suppose that a non-EU cloud provider only stores the encrypted data and shared parts of the key. If the data cannot be accessed without key shares stored separately within the EU (e.g., by another independent EU cloud provider), this increases the trust of the user from the EU towards the non-EU cloud provider. In the future, the EU is expected to adopt the ePrivacy Regulation (ePR) [3]. Complementing the original GDPR, this regulation includes stronger rules that regulate the confidentiality of electronic communications.

The main aim of this article is to explore the practical use of secret sharing to protect private user data. In the first part, we give an overview of secret sharing schemes. We give more details of Shamir's secret sharing scheme that is useful for situations when we need to split the secret into n shares and allow reconstruction if at least t shares are present.

The next section focuses on the review of existing solutions that employ secret sharing in practice. We then describe our solution for the secret sharing of private data on iOS phones using common commercial clouds. In the final section, we discuss potential applications of secret sharing for securing IoT devices.

2. Secret Sharing Schemes

Informally, a secret sharing scheme is a way to split a secret within a group of participants. Our goal is that some selected groups of participants can reconstruct the original secret, while others are not able to learn any information about the secret.

Formally, let P denote a set of users. The power set of P denoted by $\mathcal{P} = 2^P$, is divided into two parts. The qualified sets from $\mathcal{P}$ form an access structure $\mathcal{A} \subset \mathcal{P}$. For a practical secret sharing scheme, we consider monotone access structures. An access structure is monotone if, for any $A \in \mathcal{A}$, each superset B of A (i.e., $A \subset B$), is also in the access structure, $B \in \mathcal{A}$.

Let $\mathcal{K}$ be a key space that is a set of possible secrets (keys) that we want to share between the participants in P. Each key is typically represented as a bitstring, but in some protocols, it can be understood differently, e.g., as a binary representation of some large integers. Let $\mathcal{S}$ be a space of key shares. Key shares typically have a similar encoding as keys, but, in general, the key share space can be different from the key space.

The secret sharing scheme is defined in [4] by a set of distribution rules. A distribution rule is a function $f : P \to \mathcal{S}$, with $f(P_i)$ describing a share that the participant P_i gets. For each key $k \in \mathcal{K}$, $\mathcal{F}_k$ denotes a (publicly known) set of distribution rules for sharing the key k. The secret sharing scheme is then a set

$$\mathcal{F} = \bigcup_{k \in \mathcal{K}} \mathcal{F}_k.$$

A general definition can sometimes be too complex for understanding in the context of practical applications. To explain the secret sharing in simpler terms, we can introduce it as a protocol with a trusted dealer D, and a set of n parties $P = \{P_1, P_2, \ldots, P_n\}$. In practical applications, the dealer is typically some trusted server or a specific trusted service. The set of parties P aggregates a set of client devices, or a particular set of services that require access to some (shared) secret.

At the beginning of the protocol, the dealer knows (or generates) a new secret k. Dealer uses his secret sharing algorithm *Share* to compute n shares $s_i = f(P_i)$. The function f is stochastic and chosen from the set $\mathcal{F}_k$. While each $\mathcal{F}_k$ is public, an attacker does not know which set of rules from $\mathcal{F}$ was used (because he does not know k). Similarly, only the dealer knows which function f from $\mathcal{F}_k$ was used. In practice, this typically requires a one-time secret that is discarded after shares are distributed.

The power set $\mathcal{P}$ represents all possible groups of participants. Qualified sets represent groups of participants who are allowed access to some secret k if they all cooperate in the protocol. In practice, this is enabled by running some distributed algorithm *Reconstruct* with the individual shares given as an input. It is also possible to imagine another trusted

service that performs the *Reconstruct* algorithm on behalf of the group. If the shares of a qualified set of participants are provided to *Reconstruct*, the algorithm should output the original secret k. On the other hand, any group of participants who do not form a qualified set should not learn any information about the secret k from their shares.

Secret sharing schemes can be quite complex, depending on the requested access structure.

In our research, we focus on privacy. Private data typically belong to a single individual party, and thus there is a question of the applicability of secret sharing schemes designed for multiparty scenarios. This research is occasioned by the fact that it is very difficult for an individual user to properly care for his private data in the increased threat landscape. Instead, the user relies on some external entities to store her private data. The set of these entities can represent the set of participants, and the user (or some trusted service working on her behalf) plays the role of the dealer (and runs the *Reconstruct* algorithm).

A general form of secret sharing with generic access structures allows us to differentiate the parties and their groups. In the case of simple secret sharing with equivalent parties, we can use a simpler model with a threshold secret sharing scheme. A threshold scheme is defined by an extra parameter t. Any group of t or more participants (out of total n) is qualified, and any smaller group of participants is not qualified. We can use the following definition:

Definition 1 ([4]). *Let $t, n \in \mathbb{Z}^+$. A (t, n)-threshold (secret sharing) scheme, with $t \leq n$, is a method that divides the original secret k into n shares s_i, $1 \leq i \leq n$ distributed for n participants $P_i \in \mathcal{P}$. Any group of t and more participants can reconstruct k, but any group with $t-1$ or fewer participants cannot learn any information about k.*

This means that a set $A \in \mathcal{P}$ is qualified if and only if $|A| \geq t$. In practice, using a threshold scheme means that given some n available storage services, we can select some confidence parameter t and allow access to the data only for groups of at least t storage services.

Shamir's Secret Sharing Scheme

Shamir's secret sharing scheme is one of the first threshold schemes, formulated by Adi Shamir in 1979 [5]. The scheme is based on polynomial interpolation. Any polynomial $y = f(x)$ of degree $t-1$ is uniquely defined by t distinct points (x_i, y_i). The Dealer chooses a suitable finite field polynomial and shares n distinct points on the graph of the polynomial with n users. Any group of at least t users can then reconstruct the polynomial.

The (t, n) Shamir's secret sharing scheme can be defined as follows:

1. Dealer D generates a secret S, with $S \geq 0$.
2. D chooses a prime p such that $p > \max(S, n)$.
3. D assigns the secret value as $a_0 = S$.
4. D randomly generates other $t-1$ coefficients $a_1, \ldots, a_{t-1}$, $0 \leq a_j < p$. The coefficients define a random polynomial f over the finite field $\mathbb{F}_p$, $f(x) = \sum_{j=0}^{t-1} a_j x^j$.
5. D computes the shares as $S_i = f(i) \mod p$, for $1 \leq i \leq n$
6. D sends (in a secure way) a share (i, S_i) to each participant P_i.

To reconstruct the secret, at least t points $(x_i, y_i) = (i, S_i)$ are needed. Suppose that some subgroup of t participants has collected their shares. It is then possible to compute coefficients a_j of the polynomial f by Lagrange interpolation:

$$f(x) = \sum_{i=1}^{t} y_i \prod_{1 \leq j \leq t, j \neq i} \frac{x - x_j}{x_i - x_j}$$

After computing the polynomial f, participants can calculate the shared secret S using the fact that $S = a_0 = f(0)$. Thus:

$$S = \sum_{i=1}^{t} c_i y_i, \qquad \text{where } c_i = \prod_{1 \le j \le t, j \neq i} \frac{x_j}{x_j - x_i}$$

Shamir's secret sharing scheme is perfectly secure [5]. If the group of participants that wants to reconstruct the shared secret has fewer than t participants, the interpolation fails, and the smaller group does not obtain knowledge of the shared secret. Note that in practice this only holds if the polynomials are always uniformly randomly chosen and are not reused when sharing multiple secrets.

3. Practical Use of Secret Sharing

Secret sharing has many applications in a modern distributed computing environment. In this section, we provide a review of proposed solutions that employ secret sharing techniques. We can roughly divide those into two groups, cloud data storage and data sharing and aggregation from IoT devices.

The first area that comes into focus where secret sharing may be used is cloud data storage. It ensures data confidentiality, but also redundancy, when using a threshold scheme with appropriate parameters.

Buchanan et al. [6] present a cloud data storage technique using secret sharing focused on features such as redundancy or break-glass data recovery. In addition to Shamir's secret sharing, the authors evaluate two other secret sharing schemes and compare their performance.

Loruenser et al. [7] outline secure cloud-based data sharing by combining Byzantine fault-tolerant techniques with secret sharing. The work is focused on sharing without any single point of trust or failure.

Kim and Lee [8] propose an enhancement of the Hadoop cloud storage system in the form of Pinch's online secret sharing scheme, which is used to distribute a secret key in multiple fragments. Thanks to this, exposure of block access token information and man-in-the-middle attacks are prevented.

Le et al. [9] apply a secret sharing scheme to the problem of medical database storage. Encrypted patient records are partitioned using the secret sharing algorithm, and each fragment is saved to a different cloud service provider.

Naz et al. [10] present a data sharing platform using various technologies, including a blockchain, an interplanetary file system, and secret sharing. Shamir's secret sharing scheme is used to divide file hash into multiple shares.

Singh et al. [11] present a secure data deduplication scheme using various secret sharing techniques. The permutation-ordered binary number system is used to distribute data into secret shares, and the secret sharing based on the Chinese remainder theorem is used to distribute key shares among key management servers.

One of the security threats in IoT networks is malicious nodes that sniff network traffic and may interfere with node activity by sending erroneous packets. Secret sharing is one of the techniques that can prevent malicious activity and even improve the performance of the network. Below are a few examples from the current literature that use various kinds of secret sharing.

Fu et al. [12] propose a non-interactive secure multidimensional data aggregation scheme. An additive secret sharing scheme is used, where individual private data entries from mobile users are split into two parts, where each part is sent to a different cloud server. These data are then aggregated on the cloud side and sent to a control center for analysis.

Cha et al. [13] define the requirements of secure communication in a smart city. These are privacy, integrity, efficiency, scalability, and decentralization. The solution to the data secrecy problem includes distributing private data to several cloud service providers using secret sharing schemes.

Chen et al. [14] focus on securing wireless sensor networks using a light-weighted secret sharing scheme. The original message is split into multiple shares and sent using different routes to the sink node.

Kamel et al. [15] present a decentralized resource discovery model in an IoT network. Additive secret sharing is used to distribute heavy computations among network nodes.

Rehman et al. [16] present an energy-efficient IoT e-health model where homomorphic secret sharing is used by network edges for the generation and distribution of secrets among nodes.

In Salim et al. [17], secret sharing is used for the selection of virtual edge nodes, so that untrusted cloud servers are not able to deduce which computations were performed.

From the abovementioned papers, we can see that current research is focused on complex distributed systems with a large number of users (smart city, enterprise cloud solutions such as e-health). Large companies that have a great demand for the cloud and distributed computation can benefit greatly from these solutions. What is missing is a simple solution that focuses on personal usage of the cloud for the protection of private data. This solution should be accessible to anyone with a suitable mobile device. This paper presents such a solution.

4. Datachest: An Application of Secret Sharing for Privacy in Clouds

Datachest is an application developed for the iOS platform. It targets the ordinary user who does not have advanced knowledge in information technology. Therefore, the application is user-friendly and has a simple and intuitive interface while keeping all complicated processes in the background. The user is offered an additional layer of security (secret sharing) while managing their confidential files. The main purpose of Datachest is to upload users' data to commercial cloud storage while utilizing secret sharing so that no cloud provider can read users' data in the cloud. All implementation details are described in [18] and the implementation is available in [19].

The Datachest application uses a threshold secret sharing scheme with parameters $n = 3, t = 2$. This means that a secret is always split into 3 shares, and at least two are needed to recover a given secret. We have chosen 3 commercial cloud providers for the general public, i.e., Google Drive, Microsoft OneDrive, and Dropbox. These are the top 3 cloud storages that are the most used at present, according to public surveys [20,21]. The selected companies are also competing in the business sphere, which strengthens our security model. We suppose that their cooperation with the purpose of (unlawful) compromise of their users' data is unlikely, as such cooperation would negatively impact their business. Note that the current situation might be different when considering commercial cloud providers; see, e.g., [22]. It is possible to adapt the solution to a different set of cloud providers due to its modular architecture.

4.1. Conceptual Architecture

Conceptually, the architecture of Datachest is relatively simple. The user interacts primarily with the application. They can upload or download their confidential files, see a list of uploaded files, and manage access to each cloud provider. Datachest uses the APIs of each cloud provider to communicate with the specific cloud. One of the disadvantages of this approach is that the application needs to be maintained regularly due to possible API changes or deprecation of various endpoints. Our architecture is modular and scalable to support API changes and possibly more cloud providers in the future.

The Shamir's secret sharing used in the application is based on the open source library [23]. This scheme (and its implementation) is not sufficient performance-wise when using larger files as input. Thus, in practice, we needed a two-step process. Private data are first protected by standard symmetric encryption, which is fast and simple to use. The data should remain secret as long as we keep the key secret, which is a short 128-bit string. Therefore, the problem of confidentiality is delegated to the protection of the key.

In the second step, the secret key that can be used to decrypt the data is stored in the cloud using a secret sharing scheme. With this change of architecture, it is possible to retain acceptable performance since the secret sharing algorithm operates only on small inputs: the cryptographic keys.

This process is summarized in Figure 1. The file is first encrypted using the AES cipher in GCM mode with a 128-bit key, as recommended by current encryption standards (such as TLS [24]). Note that the AES algorithm can be changed to another secure encryption algorithm if there are some concerns about AES security, such as [25,26]. The encrypted file is uploaded to the cloud storage the user chooses. The key is split into 3 shares using Shamir's secret sharing algorithm. Each share is uploaded to a different cloud storage. Finally, the cryptographic key on the mobile device is destroyed.

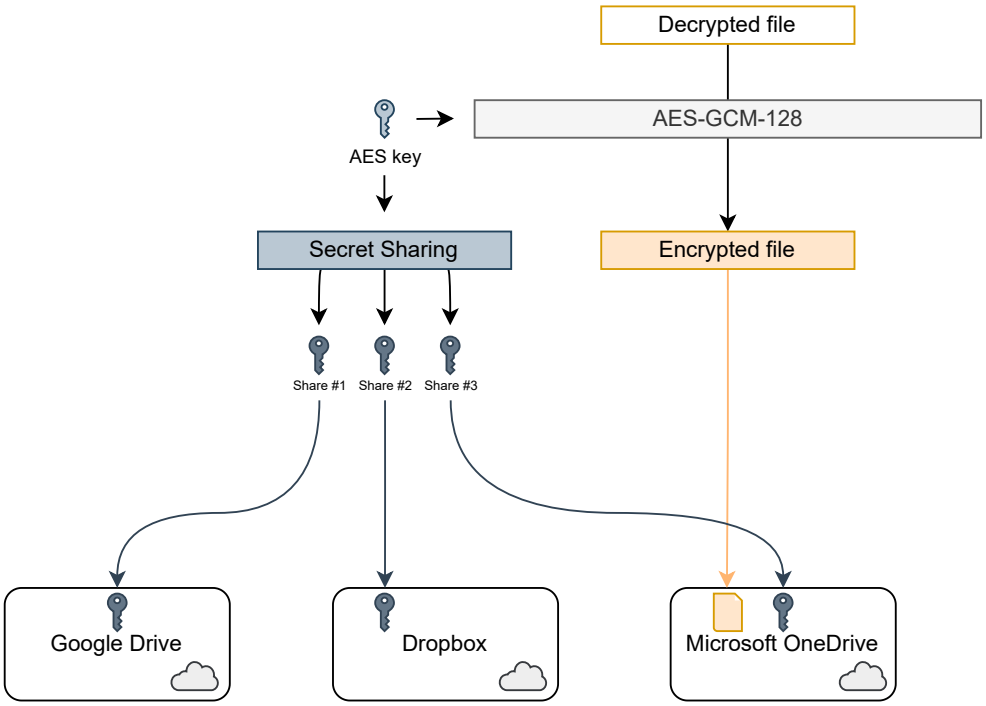

Figure 1. File uploading scheme in the Datachest application.

4.2. Design Architecture

From the point of software implementation, Datachest follows some design patterns which we will discuss in more detail.

4.2.1. MVVM

The core design pattern is MVVM. It is an acronym that stands for Model, View, and ViewModel. MVVM is widely used in mobile development. The Model represents data and their structure. Every piece of information in the application is structured into data models. View represents the graphical components that build individual screens and eventually the user interface. A View can be a button, form, a dropdown menu, or a whole screen built with smaller Views. Views should be primitive, meaning they should not contain any logic, which should be the ViewModel's responsibility. The ViewModel is a class belonging to a specific View. Each View is assigned to one and only one ViewModel, and vice versa. The ViewModel provides View with data structured in Models and logic. For example, when a user clicks on a button in a View, the View emits an event to its ViewModel and the ViewModel handles that event in a series of actions. In a hierarchy, the Model feeds the ViewModel with data, and the ViewModel feeds the View with received data and handles events that occur in the View.

In conclusion, the overall purpose of an MVVM design pattern is to separate data, logic, and UI into meaningful, reusable layers. Thus, the application can scale effectively, and the code is easily organized.

4.2.2. Application Store

Data that are not isolated to a single ViewModel are present in a so-called Application Store. It is a component that keeps this kind of data in one place, available to all parts of the application. The Store is reactive, meaning that every change in the Store is immediately reflected in the UI. This behavior is useful when notifying users in case of an error or when a user signs in. The Application Store contains data such as access tokens of a signed-in user, list of ongoing uploads or downloads, state of login, etc. The Store is 'a single source of truth'. That means that data present in the store should not be present elsewhere and that the Store should always contain data in its most recent state with full integrity.

4.2.3. The Core

We have stated that all logic specific to a View is handled by a ViewModel, assigned to a given View. However, some processes are not View-specific. Thus, we specified an application core. This is a segment where more complex and low-level logic is executed. The core mostly contains stateless singleton services. These are classes that do not have a state. Thus, they do not contain any data. These services only contain a list of methods that require inputs, provide outputs, and contain operations executed over these inputs. Classes are singletons, which means they can be initialized only once. Since they are stateless, there is no need to have more than one instance of each service. Services perform operations such as communication with cloud APIs, work with a phone's local storage, authorization operations, etc. The core also contains other types of classes, such as classes that represent upload and download sessions. These cannot be stateless, nor singleton, because they are specific for each upload or download session.

ViewModels connect with the core via so-called Facades. Facades are classes with a list of methods that a ViewModel can call. These methods only forward requests from ViewModels to the core and format received inputs beforehand, if necessary.

In the end, we can simplify the software architecture into 3 parts: data, logic, and UI. The UI part consists of Views. The logic part consists of ViewModels, Facades, and the core. And finally, the data part consists of Store, external data coming from various sources, and data models, which define a structure into which every piece of information should be formatted. The hierarchy is summarized in Figure 2.

4.3. Authentication

Signing in and out, authorization and verification are handled by the OAuth 2.0 protocol [27]. A user can log into his Google, Microsoft, and Dropbox accounts and grant Datachest access to perform some authorized operations on behalf of the user. These operations include reading information about the user's profile, uploading, downloading data, and reading information about uploaded data. Each company provides an open source authentication library to simplify implementing authentication mechanisms. We used these libraries when implementing Datachest. They handle the whole sign-in and sign-out process in the background and also display the UI for the user to sign in or out. The main output of a successful sign-in flow is an access token. The token is a string that is used to perform authorized operations on behalf of the user. In Datachest, access tokens are required to perform API requests (that is, uploading, downloading, reading files, etc.). They are sent in the authorization header of every HTTP request between Datachest and cloud APIs. After a token has successfully been obtained from the sign-in flow, it is saved to the Store, where it resides during the application runtime. We also save tokens and their expiration dates to a secure encrypted device database, "Apple Keychain" [28]. Then, at launch, the application searches for the user's persisted access tokens in the keychain. In case the tokens are found and still valid, a 'silent sign-in' is performed. That means that

Datachest will try to sign the user in without their interaction so that they can perform all authorized processes right after the application launches. In case tokens are not found or are expired, Datachest is unable to sign the user in silently, and thus, the user has to sign in manually.

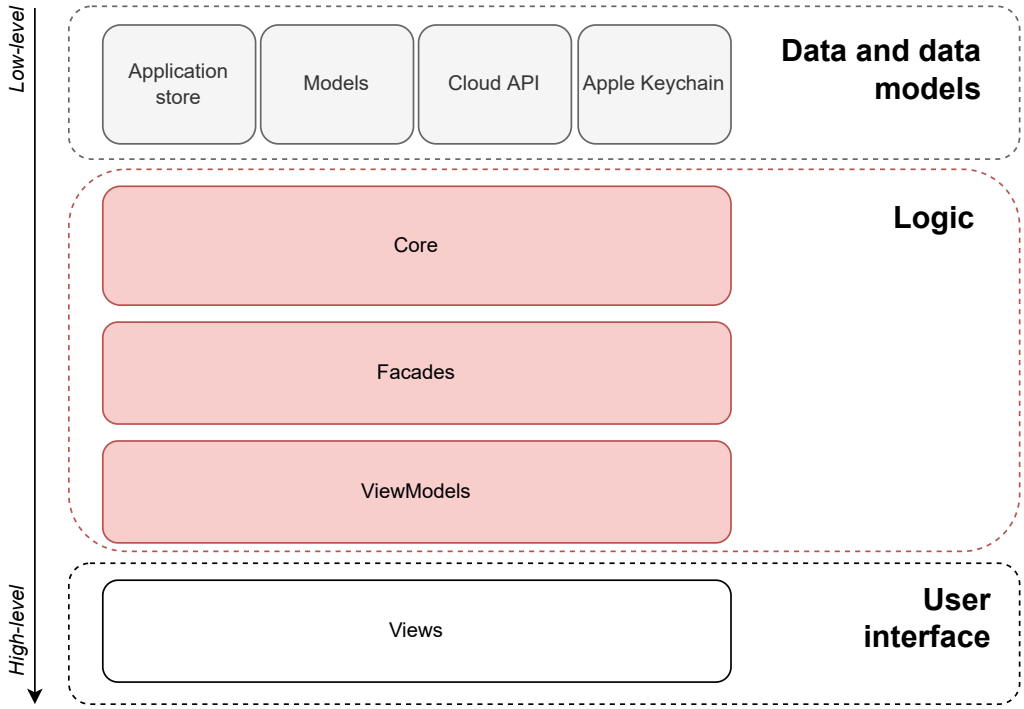

Figure 2. Hierarchy of components of the software architecture of Datachest.

4.4. Firestore

As shown in Figure 1, every file is uploaded in encrypted form and its corresponding key shares are uploaded to every cloud. This brings up the problem of pairing shares back together and also pairing shares to a corresponding file. To track evidence for this, we use Firebase [29]. Firebase is a service from Google that helps satisfy the back-end needs of front-end applications, without the need to develop and deploy an actual back-end application. We specifically used Firestore, which is a Firebase service of a NoSQL database. Then, when Datachest finishes uploading a file, it creates a 'document'. The document only contains the IDs of each file, i.e., the ID of the uploaded file and IDs of individual shares of a key that correspond to the file. It is then easy for Datachest to download a file from the cloud, find its corresponding key shares, and reconstruct the file using secret sharing and decryption.

4.5. File Management on Cloud and Mobile Device

Each processed file is saved to the cloud (upload) or to a mobile device's storage (download). There is a root folder called *Datachest*. The purpose is to isolate the application data from the rest. There are two subfolders under *Datachest*, specifically *Files* and *Keys*. *Files* contain all encrypted (cloud) or decrypted (mobile device) files processed by the application. *Keys* contain key shares that were used to encrypt all files. This subfolder is also important during the download and decryption process. It is present only in the cloud for security reasons. The application creates these folder structures if they do not exist.

4.6. Uploading Files

This process was previously briefly mentioned in Section 4.1 and visualized in Figure 2. Reading the file from the phone's local storage and encrypting it at the same time brings a threat of memory overflow. Since the user is not limited in size when choosing a file to

upload, Datachest has to deal with this problem in order to prevent it. Thus, we process the file by inserting it into a stream as shown in Figure 3. We read the file in chunks of specified size. Each chunk is encrypted individually and uploaded to the cloud afterward. We use one cryptographic key and one nonce per file. One authentication tag is returned from the cipher per chunk. That results in an array of tags after a file has been processed in most cases. After all the chunks have been uploaded, the stream is closed. This also makes it easier to track the file upload progress, which is useful when showing the user the file upload progress. Cloud APIs also support this approach. For example, Google Drive API calls this method a 'resumable upload' [30]. This means that the file is uploaded in chunks, sequentially. This brings another advantage, in that uploads can be interrupted and resumed later without having to start over.

After a file upload is completed, the cryptographic key is split into three shares. We then create three files. Each file contains a key share. Additionally, one of the files also contains other cryptographic information, such as what nonce and authentication tags were present during the encryption. This information is also necessary for a successful decryption, since we are using GCM mode. After this, we create a document at Firestore, write individual file IDs to it, and create their respective association.

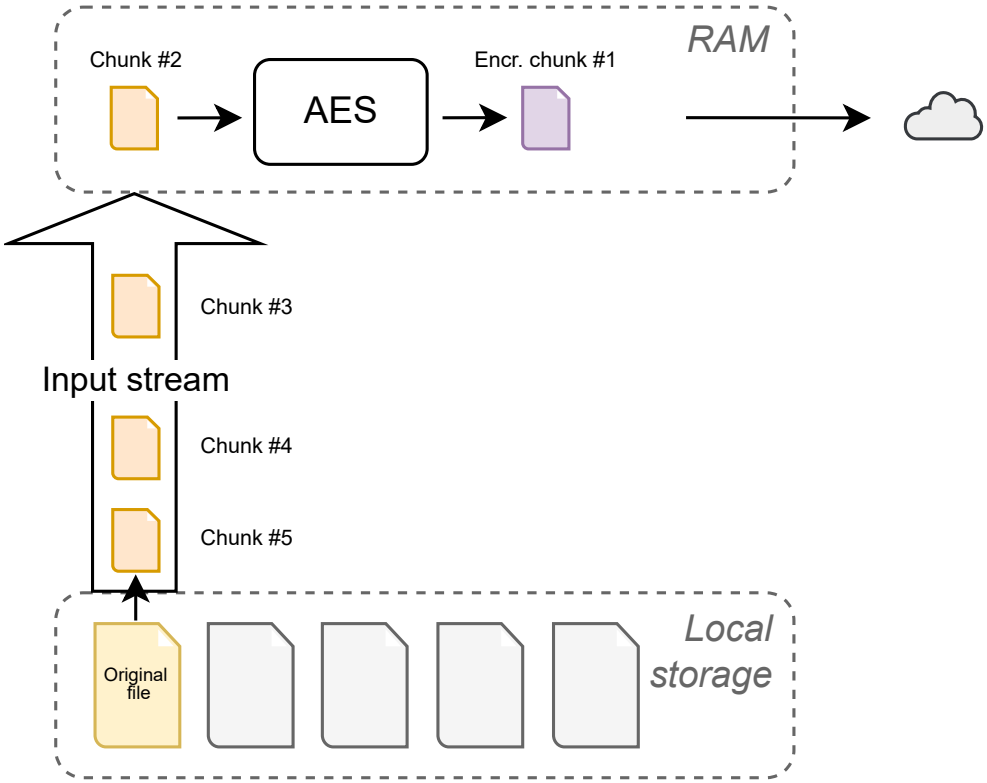

Figure 3. Encrypting a file via streams in the Datachest application.

4.7. Downloading Files

After a user chooses a file they would like to download, the process starts with downloading key share files from individual cloud storages. If a user is only signed in to two out of three cloud providers, it is still possible to fulfill their request, since our Shamir's secret sharing scheme allows a threshold of $t = 2$. Before downloading these files, Datachest calculates which files to download by acknowledging the information from Firestore. Datachest sends Firestore the requested file's ID, and Firestore returns IDs of all key share files.

After all needed key share files are downloaded, the main file is downloaded. Using Apple's native API in the Swift programming language (in which this application is written), a downloaded file is persisted to a temporary storage location [31]. A programmer can

work with the file reference, which is returned as a result of the download process. If the reference is not used, the file is automatically deleted from the temporary location.

The next step is to use the AES cipher to decrypt the downloaded file. First, key shares are combined using Shamir's secret sharing algorithm into the former key. Secondly, we use the obtained key along with correct nonce and authentication tags to decrypt the file. The file is now being processed chunk by chunk through streams sequentially. The size of the chunk has to be the same as when the file was uploaded. This rule has to be followed due to authentication tags. Since every authentication tag belongs to a specific chunk, redefining its size would result in an authentication error. A pair of streams is opened to process the downloaded file. The first stream is an input stream that connects the temporary storage location with the phone's memory. Each chunk is processed in memory, where decryption takes place. A decrypted chunk is sent to the second stream, which is the output stream. This stream connects the phone's memory to a new, permanent storage location, defined by Datachest. The decryption process can be seen in Figure 4.

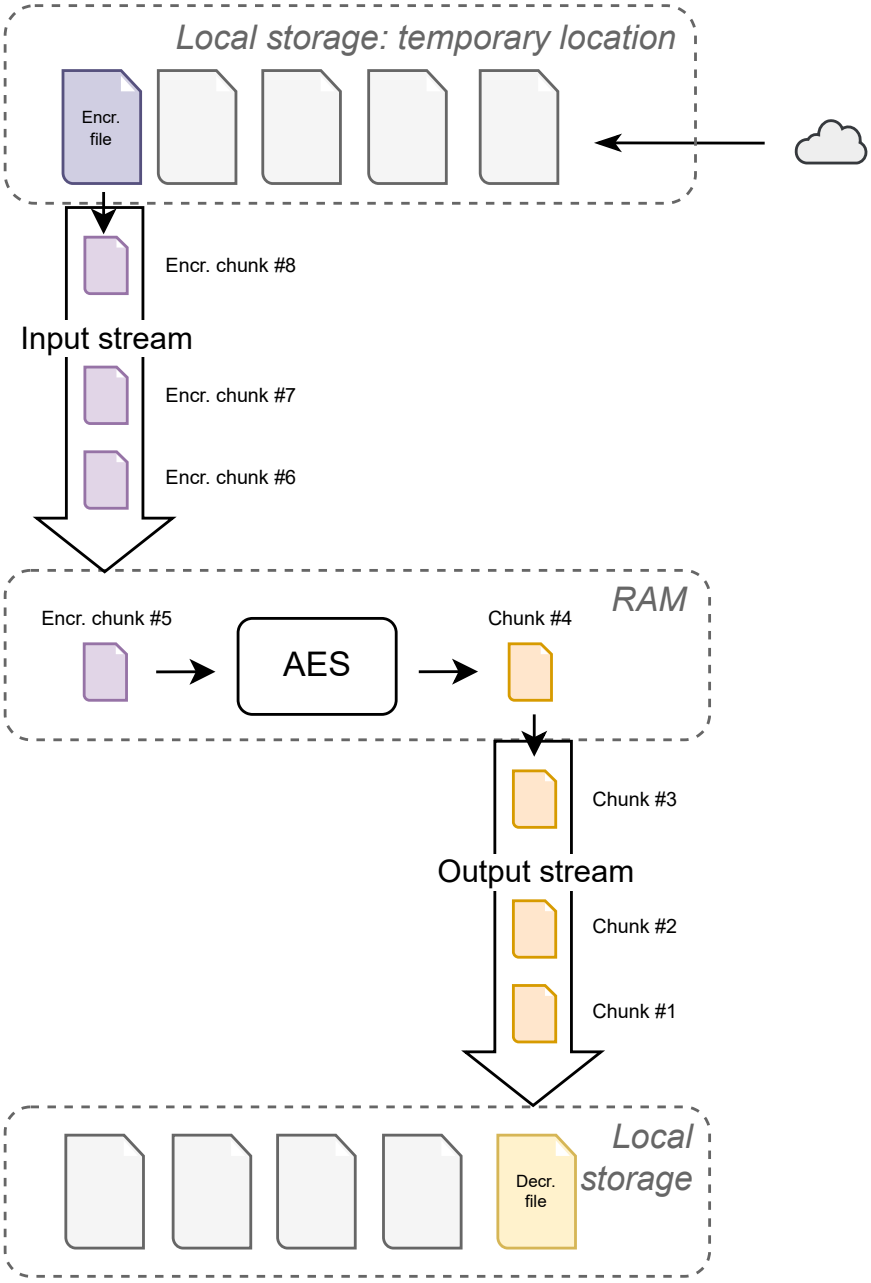

Figure 4. Decrypting a file via streams in the Datachest application.

4.8. Performance Evaluation

The performance of the Datachest application depends on several factors:

1. **Network.** Network connection quality is crucial when uploading or downloading files. Unstable or poor connection leads to unsatisfactory performance. Network connectivity is also required for communicating with APIs (authentication, fetching various information about user files).
2. **Performance of external components.** Although statements in the previous point are true, they are not sufficient in discussion of the application's performance over the network. Responsive and fast communication with APIs also depends on their performance. Availability and fast server response times are expected. This factor is outside Datachest's area of influence.
3. **Encryption.** The impact of encryption is negligible in terms of performance. Encryption with AES is fast and effective. An encryption performance test has been executed on a file with a size of 1 MB. The application has been able to encrypt the file with a satisfactory speed of 0.004 s [18]. Larger files do not cause a significant drop in encryption performance.
4. **Secret sharing.** Secret sharing is a costly operation, performance-wise. Sharing a file with size of 1 MB takes more than 30 s in the Datachest application [18]. That is why only cryptographic keys are used as inputs to this algorithm (see Section 4.1). Their size is exactly 128 bits, which is acceptable in terms of performance.

5. Applications of Secret Sharing for Securing IoT

The International Telecommunication Union (ITU) defines the Internet of Things (IoT) in Recommendation Y.4000/Y.2060 as:

> A global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) things based on existing and evolving interoperable information and communication technologies [32].

The IoT generalizes the standard Internet architecture and enables interconnection between the physical world and cyberspace. In our recent survey [33], we have shown that there are still many security challenges for the IoT. In the following section, we will focus on two main security goals, namely, preserving the privacy (and possibly also integrity) of the data while also keeping the data available. To achieve these goals, we can use the tools provided by secret sharing.

In Section 4 we have shown details of the Datachest application that uses secret sharing among multiple cloud storage providers to secure the private data of users on a mobile device (a smartphone). In this section, we would like to generalize these results and provide new architectural models that can enhance IoT security by using secret sharing.

5.1. Network and Connectivity Model

The IoT represents a general Internet architecture. In our research, we focus only on the application layer of the network stack. Thus, we have the following prerequisites for the application of secret sharing on the level of network and connectivity:

1. We understand the network as secure interconnection of communicating nodes (see Section 5.2 for more details on node types). Connection between nodes is typically secured with the TLS protocol or other standard secure network encryption.
2. The network handles all transport security and authentication.
3. If the IoT device does not have a full network stack implemented, we assume that there is a controller or proxy node that will become a participant as a proxy for the lightweight node (or multiple lightweight nodes).

To simplify the situation, we do not consider the specifics of the IoT, such as the concrete capabilities and limitations of IoT devices. Furthermore, we do not consider exact location of data and data processing limitations, such as analysing Big Data as close as possible to the Edge. Our secret sharing model assumes a concrete point at which data

are ready for storage in the cloud. However, this point can be an abstraction for a whole underlying IoT network that produced data that we want to further protect in cloud storage. For private data processing, more complicated solutions are needed, such as garbled circuits or homomorphic encryption [34].

5.2. Participants and Data Structure

When considering applications of secret sharing, we first need to correctly identify the set of participants and the corresponding access structure. We also need to consider the role of the Dealer as a trusted service that will process our secret data into shares. Finally, we need to provide the reconstruction service that will allow participants to combine their shares and reconstruct the original secret data.

In the IoT context, we focus principally on 3 types of entities:

1. **IoT devices.** These are the physical and virtual things connected to the IoT as end nodes. These devices can produce and consume data. Devices vary in their processing and storage capacity from small RFID chips to full computers/smart devices. From a security perspective, these devices can be especially vulnerable due to their physical location and limited resources that can be dedicated to security management. The IoT user typically interacts with the devices directly and is responsible for them.
2. **Cloud storage.** There are various uses for cloud services in the context of IoT. We will separate one special category: cloud storage providers. We understand them as services that enable persistent distributed storage of data with an API that enables read and write access after some initial authorization. From a security perspective, cloud storage provides a high level of security. The provider typically protects the confidentiality and integrity of the data, provides backup, and tries to maintain high availability. The IoT user typically does not interact directly with the stored data and is not responsible for their management.
3. **Management of IoT.** Local or cloud services that expose APIs available for multiple IoT devices. These can handle configuration data for the devices, provide authorization tokens, collect and aggregate data from the devices, and do various other tasks. Management services can be under the control of the IoT owner or manufacturer, or leased as a special cloud service. The security depends on the exact scenario and the use of the management services, but typically these services must be very secure. A compromise of the management service can typically compromise multiple devices on the IoT.

5.3. Threat Model

Because our network and connectivity model assumes that the network handles transport security and authentication, our threat model is unique for IoT applications. This allows us to focus on higher-level threats and abstract the main network threats. On the other hand, if the underlying network layer is compromised, our solution will be broken as well.

Furthermore, we add another security assumption, i.e., that the cryptographic primitives used are secure and correctly implemented in the corresponding library. Similarly to the network layer, this allows us to abstract cryptographic threats at the cost of possible breaks on this layer.

The main threats we want to solve with the help of secret sharing are the following two categories:

1. Information Disclosure. While we suppose that the data is secure during transit thanks to network security, it can still be compromised in storage, both in local storage on the IoT device (e.g., by physically obtaining the device), or in remote storage on cloud (e.g., by compromising access via phishing).
2. Denial of Service. Attacker can prevent the access to data in remote storage, e.g., by executing Distributed Denial of Service, or simply by deleting the data after compromising the access.

Our security model is based on the following premises:

1. IoT devices can be compromised; we do not want to store long-term secrets on them. They can also fail, limiting the availability of stored data.
2. We model IoT management as a trusted service with high availability, suitable for security critical tasks. However, we suppose that it has only limited storage capacity (storage is delegated to cloud storage services), so we use it only to store authorization tokens.
3. Each cloud storage has some probability of data compromise (this covers both information disclosure and tampering) p_c. This can be caused by a malicious internal actor or by unauthorized access (e.g., after a phishing attack). We suppose that the compromises of different cloud services are independent.
4. Each cloud storage has some probability of denial of service (DoS) p_d. Note that this includes all types of DoS attacks, including distributed and self-inflicted DoS. The probability p_d thus includes all availability issues regardless of their source. Again, we assume that availability compromises of different cloud services are independent.
5. A special case is a loss of connection of some internal IoT subnetwork to all external cloud storage providers at once, which we denote as an offline state.

A suitable application of a secret sharing scheme can improve both resistance against data compromise and DoS attacks. We will now consider specific architectures that employ a simple threshold secret sharing scheme.

5.4. IoT Device Based Secret Sharing

Consider a smart IoT device with full computing capabilities comparable to a typical smartphone. In this case, we can employ a (generalized) version of the Datachest solution to protect the data.

The Datachest application handles the connection to clouds via OAuth. This task can be delegated to the IoT management service, or provided directly on the smart IoT device, depending on the security management of the overall system.

In this architecture, secret sharing and reconstruction is handled on the device. Small files can be secret-shared directly, but for efficiency reasons it is suitable to include encryption/decryption on the device as well (similar to Datachest). The dealer application distributes shares (and encrypted files) among cloud storage providers. When the IoT device needs to read the data, it collects the required shares (t shares in a threshold scheme) from the cloud storage and reconstructs the data according to the scheme.

Let us suppose that we use a (t, n)-threshold secret sharing scheme with n cloud storage providers. To compromise the data, the attacker would have to compromise t out of n services. The probability of compromise in our security model with independent clouds becomes p_c^t. We can increase data security by increasing the parameter t.

To access the data, the IoT device needs to access at least t clouds. In case of DoS, the probability of blocking access to the required shares becomes p_d^{n-t+1}. To increase availability, we need to reserve a suitable number of additional storage services above the threshold t.

Increasing the parameters t, n increases the demands on storage space, computation of secret shares, and data communication during both read and write operations. The Datachest application solves this by using symmetric encryption and applying secret sharing only on the (small random) keys. When using secure encryption, this preserves data security and its scaling with the parameter t. However, data availability is limited and the probability of losing access to data becomes fixed to p_d (since the encrypted file is stored in a single cloud storage).

To increase availability in the model with encrypted data and secret shared keys, we can dedicate one extra cloud storage with higher availability (DoS chance p'_d) for encrypted files and use other cloud services (with lower availability) just to store secret shared keys. To achieve the same availability, we should have $p'_d = p_d^{n-t+1}$.

5.5. IoT Management Based Secret Sharing

The first architecture analyzed in the previous section requires a smart IoT device that can handle the secret sharing algorithm, reconstruction algorithm, and optionally also encryption and decryption. For limited devices, all these tasks can be delegated to an IoT management service. The architecture is again similar to the Datachest application, but both the Dealer and Reconstruct algorithms (and optionally encryption/decryption) become parts of the trusted IoT management services. The IoT device needs a secure connection to these services.

The main advantage of delegating the secret sharing tasks to a management server is the flexibility of the architecture. The management server can handle the data security and availability for multiple IoT devices in a transparent way. The main disadvantage is that we rely on a single point of failure. A compromise of the management server leads to a compromise of the whole IoT subnetwork managed by this server. However, a centralized security node might be easier to secure than securing multiple individual IoT nodes. Moreover, the data stored in cloud storage (protected by secret sharing) are not compromised even when the management server is, unless (at least t) access tokens to the corresponding storage clouds are leaked.

5.6. IoT Secret Sharing with Dedicated (Local) Storage

The previous two architectures were a generalization of our Datachest architecture with n cloud storage providers having the role of secret sharing scheme participants. Let us define an IoT-specific architecture where we assign different roles for different storage types. We will use a $(2, 3)$-threshold scheme with file encryption. The three participants are:

1. small local storage on the IoT device, just for key shares;
2. large cloud storage for key shares and encrypted files;
3. dedicated (local) storage (backup/fallback) for key shares (and possibly encrypted files if we want to enable offline mode).

Dedicated (local) storage should be connected to the IoT management service and independent of normal cloud storage. Cloud authorization, encryption/decryption, secret sharing, and reconstruction should be handled by the IoT management service. Reconstruction can also be delegated directly to the device if it has the corresponding computing capabilities.

We can use a similar mechanism as in Datachest for protecting the data. When storing files, each file is encrypted with a random symmetric key, and the key is secret-shared. The key shares are distributed to each of the participants (and kept in the corresponding storage). The encrypted file is stored in the dedicated storage cloud and can also be duplicated in the dedicated local storage.

When reading data, we define 3 modes:

1. *Normal mode*: The encrypted file is accessed in the cloud together with the first key share. The second key share is obtained from the IoT device. Two shares are used to reconstruct the key and decrypt the data.
2. *Backup mode*: If the IoT device is compromised or inaccessible, the IoT manager or data owner can still reconstruct the key that can be used to decrypt the file stored in the cloud by accessing a key share from the dedicated local storage.
3. *Offline mode*: If the encrypted file is mirrored to the dedicated local storage, the IoT device can work in an offline mode without access to the cloud storage. The dedicated local storage is used to get the encrypted file and one of the key shares, instead of a cloud.

An offline mode can be implemented in a partial way for data that require high availability (such as basic configuration data). Dedicated local storage can also be replaced by independent cloud storage, basically acting as a backup/fallback system for the normally used cloud storage.

5.7. Security Analysis

The security of the proposed system is based on existing secure components:

1. Authenticated encryption with AES in GCM mode. This protects the privacy and integrity of the data blob as long as we keep the key secret.
2. Shamir's secret sharing scheme. This is used to split the secret key into shares. We assume that Shamir's (t, n)-threshold scheme guarantees that the original key is kept secret as long as less than t shares are obtained by the adversary.

Formally, let m be a message we want to protect while using a secret sharing scheme with the cloud and local storage. We can abstract the details and consider that we have n participants $\{P_1, P_2, \ldots, P_n\}$. We also have a trusted dealer D with an existing secure connection to each participant. If we want to store the secret, we perform the following protocol:

1. The dealer generates a random 128-bit key k.
2. The dealer encrypts the message m with AES-GCM and obtains the encrypted message c.
3. The dealer sends the encrypted message c to (in general) e (out of n) participants for long-term storage.
4. The dealer runs the *Share* algorithm of Shamir's (t, n)-threshold secrets sharing scheme on k and obtains n shares $s_1, \ldots, s_n$.
5. The dealer sends each s_i to participant P_i for long-term storage.
6. The dealer safely deletes the original message m and the random key k.

To reconstruct the secret message, we run the following protocol:

1. Dealer D obtains the encrypted message c from any available participant's storage.
2. The dealer obtains at least t out of n shares s_i from the participants.
3. The dealer runs the *Reconstruct* algorithm of Shamir's (t, n)-threshold secrets sharing scheme on shares s_i and obtains the original key k.
4. The dealer uses AES-GCM decryption on c with secret key k to reconstruct the original message m (and verify its integrity).

We suppose that the attacker can obtain c and wants to reconstruct the original message m. We have the following situations:

1. We assume that AES-GCM is secure and that the original 128-bit key k was generated randomly. Thus, the probability of guessing the key is 2^{-128}, which is negligible.
2. The attacker obtains l shares s_i by compromising l participants. If $l < t$, the attacker cannot reconstruct the original key (owing to the assumptions in Shamir's secret sharing scheme).
3. Let us suppose that the attacker obtained $l \geq t$ shares. The attacker can reconstruct the secret key instead of the dealer and learn the message m by decrypting c. We have supposed that the probability of the compromise of each participant is p_c and these are independent of each other. Thus, the probability of compromise of the message m is at most p_c^t. As $p_c < 1$, and $t > 1$, then $p_c^t < p_c$, and we have reduced the security risks related to the compromise of long-term storage by a factor of p_c^{t-1}.

Similarly, we can analyse the attacker's ability to block access (DoS attack) to the original message m. If we use just a single storage for the message, an attacker can block it with probability p_d. If the encrypted message is stored in e locations, the attacker can block all of them with probability p_d^e. To decrypt the message, the dealer also needs access to t shares out of n distributed among the participants. The attacker can block the decryption if he can block $n - t + 1$ participants, which can be done with probability p_d^{n-t+1}. Note that the optimal strategy for participants is to choose $e = n - t + 1$, as it minimizes the probability of success of a DoS attack, as well as storage requirements. If $e > 1$, we get $p_d^e < p_d$, and we have reduced the risks of DoS attack by a factor of p_d^{e-1}.

We assume a simple symmetrical model, where each participant is equivalent and an attacker has an equal and independent chance to compromise/block each participant. In an asymmetric scenario, security can be improved by secret sharing, if the compromise

chance of $n - t + 1$ the least secure participants is lower than the compromise chance of the most secure participant (similarly for DoS).

6. Conclusions

The secret sharing scheme allows a group of participants to share secret data in such a way that only some prescribed subgroups can reconstruct the original secret from their shares. A special case of secret sharing schemes is a (t, n)-threshold scheme, where a group of n participants share some secret and any group of at least t participants (or more) can reconstruct it.

A threshold secret sharing scheme can be applied in practice to store a secret key (or a small file) to a set of (competing) cloud storage providers. The user can reconstruct the secret information by accessing shares from at least t storage clouds. On the other hand, a compromise of at most $t - 1$ clouds does not compromise the secret data. A threshold secret sharing can also increase availability because we only need access to t from n cloud storage providers.

We have created Datachest, a proof-of-concept application that can be used to protect users' private data on a smartphone by combining symmetric encryption and a Shamir's $(2, 3)$-threshold secret sharing scheme. Secret sharing is applied to randomly generated symmetric keys, and the shares are stored in 3 different clouds. A compromise of individual clouds, either internal or caused by some attack on authentication, does not compromise the private data. Similarly, loss of connectivity to one cloud can still enable us to reconstruct the encryption key from the remaining two key shares.

The principles and architecture of Datachest can be adapted to the IoT setting, either directly on smart IoT devices or in the management layer of an IoT solution. We have also proposed a specific dedicated architecture that uses (limited) storage in IoT devices along with two external storages and a $(2, 3)$-threshold scheme to support both normal secure operations, a backup mode, and an offline mode. The proposed IoT architectures can be adapted to a wide range of IoT solutions, including smart homes, industrial networks, and other mechatronic systems.

Author Contributions: Conceptualization, P.Č. and P.Z.; methodology, P.Z.; software, P.Č.; validation, P.Č. and P.Z.; formal analysis, P.Č. and P.Z.; investigation, P.Č., R.P., and P.Z.; resources, P.Č. and P.Z.; data curation, P.Č.; writing—original draft preparation, P.Č., R.P. and P.Z.; writing—review and editing, P.Č., R.P., and P.Z.; visualization, P.Č.; supervision, P.Z.; project administration, P.Z.; funding acquisition, P.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Scientific Grant Agency of the Slovak Republic grant number APVV-19-0220.

Acknowledgments: We would like to thank the anonymous reviewers for their comments and suggestions that improved this paper's readability and content.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

AES	Advanced Encryption Standard
API	Application Programming Interface
DoS	Denial of Service
EEA	European Economic Area
ePR	ePrivacy Regulation
EU	European Union

GCM	Galois/Counter Mode
GDPR	General Data Protection Regulation
IoT	Internet of Things
ITU	International Telecommunication Union
MVVM	Model, View, and ViewModel
TLS	Transport Layer Security
UI	User Interface

References

1. Grošek, O.; Hromada, V.; Horák, P. A Cipher Based on Prefix Codes. *Sensors* **2021**, *21*, 6236. [CrossRef] [PubMed]
2. European Parliament; Council of the European Union. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation). *Off. J. Eur. Union* **2016**, *L 119*, 1–88.
3. European Comission. Proposal for a Regulation on Privacy and Electronic Communications Concerning the Respect for Private Life and the Protection of Personal Data in Electronic Communications and Repealing Directive 2002/58/EC (Regulation on Privacy and Electronic Communications). Available online: https://digital-strategy.ec.europa.eu/en/library/proposal-regulation-privacy-and-electronic-communications (accessed on 19 August 2022).
4. Stinson, D.R. *Cryptography: Theory and Practice, Third Edition (Discrete Mathematics and Its Applications)*; Chapman & Hall/CRC Taylor & Francis Group: Boca Raton, FL, USA, 2006.
5. Shamir, A. How to share a secret. *Commun. ACM* **1979**, *22*, 612–613. [CrossRef]
6. Buchanan, W.; Lanc, D.; Ukwandu, E.; Fan, L.; Russell, G.; Lo, O. The Future Internet: A World of Secret Shares. *Future Internet* **2015**, *7*, 445–464. [CrossRef]
7. Loruenser, T.; Happe, A.; Slamanig, D. ARCHISTAR: Towards Secure and Robust Cloud Based Data Sharing. In Proceedings of the 2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom), Vancouver, BC, Canada, 30 November–3 December 2015. [CrossRef]
8. Kim, S.H.; Lee, I.Y. Block Access Token Renewal Scheme Based on Secret Sharing in Apache Hadoop. *Entropy* **2014**, *16*, 4185–4198. [CrossRef]
9. Le, D.N.; Seth, B.; Dalal, S. A Hybrid Approach of Secret Sharingwith Fragmentation and Encryptionin Cloud Environment for Securingoutsourced Medical Database:a Revolutionary Approach. *J. Cyber Secur. Mobil.* **2018**, *7*, 379–408. [CrossRef]
10. Naz, M.; Al-zahrani, F.A.; Khalid, R.; Javaid, N.; Qamar, A.M.; Afzal, M.K.; Shafiq, M. A Secure Data Sharing Platform Using Blockchain and Interplanetary File System. *Sustainability* **2019**, *11*, 7054. [CrossRef]
11. Singh, P.; Agarwal, N.; Raman, B. Secure Data Deduplication Using Secret Sharing Schemes Over Cloud. *Future Gener. Comput. Syst.* **2018**, *88*, 156–167. [CrossRef]
12. Fu, Y.; Ren, Y.; Feng, G.; Zhang, X.; Qin, C. Non-Interactive and Secure Data Aggregation Scheme for Internet of Things. *Electronics* **2021**, *10*, 2464. [CrossRef]
13. Cha, J.; Singh, S.K.; Kim, T.W.; Park, J.H. Blockchain-Empowered Cloud Architecture Based on Secret Sharing for Smart City. *J. Inf. Secur. Appl.* **2021**, *57*, 102686. [CrossRef]
14. Chen, D.; Lu, W.; Xing, W.; Wang, N. An Untraceable Data Sharing Scheme in Wireless Sensor Networks. *Sensors* **2018**, *19*, 114. [CrossRef] [PubMed]
15. Kamel, M.B.M.; Yan, Y.; Ligeti, P.; Reich, C. Attred: Attribute Based Resource Discovery for Iot. *Sensors* **2021**, *21*, 4721. [CrossRef] [PubMed]
16. Rehman, A.; Saba, T.; Haseeb, K.; Marie-Sainte, S.L.; Lloret, J. Energy-Efficient Iot E-Health Using Artificial Intelligence Model With Homomorphic Secret Sharing. *Energies* **2021**, *14*, 6414. [CrossRef]
17. Salim, M.M.; Kim, I.; Doniyor, U.; Lee, C.; Park, J.H. Homomorphic Encryption Based Privacy-Preservation for Iomt. *Appl. Sci.* **2021**, *11*, 8757. [CrossRef]
18. Čuřík, P. Secret Sharing for Privacy. Master's Thesis, Slovak University of Technology in Bratislava, Bratislava, Slovakia, 2022. Available online: https://github.com/petercurikjr/datachest-ios/blob/master/Master's%20Thesis.pdf (accessed on 22 August 2022).
19. Čuřík, P. Datachest GitHub Repository. Available online: https://github.com/petercurikjr/datachest-ios (accessed on 23 July 2022).
20. Statista. Number of Consumer Cloud-Based Service Users Worldwide in 2013 and 2018. Available online: https://www.statista.com/statistics/321215/global-consumer-cloud-computing-users/ (accessed on 9 November 2021).
21. Sebastian, N. Usage & Trends of Personal Cloud Storage: GoodFirms Research. Available online: https://www.goodfirms.co/resources/personal-cloud-storage-trends (accessed on 9 November 2021).
22. Zhang, M. Top 10 Cloud Service Providers Globally in 2022. Available online: https://dgtlinfra.com/top-10-cloud-service-providers-2022/ (accessed on 22 August 2022).
23. Echo Network. SwiftySSS. Available online: https://github.com/echoprotocol/SwiftySSS (accessed on 16 July 2022).
24. Dierks, T.; Rescorla, E. *The Transport Layer Security (TLS) Protocol Version 1.2*; Technical Report. 2008. Available online: https://www.rfc-editor.org/rfc/rfc5246 (accessed on 22 August 2022).

25. Scripcariu, L.; Mătăsaru, P.D. On the substitution method of the AES algorithm. In Proceedings of the International Symposium on Signals, Circuits and Systems ISSCS2013, Iasi, Romania, 11–12 July 2013; pp. 1–4.
26. Scripcariu, L.; Diaconu, F.; Mătăsaru, P.D.; Gafencu, L. AES vulnerabilities study. In Proceedings of the 2018 10th International Conference on Electronics, Computers and Artificial Intelligence (ECAI), Iasi, Romania, 28–30 June 2018; pp. 1–4.
27. Parecki, A. OAuth 2.0. Available online: https://oauth.net/2/ (accessed on 9 April 2022).
28. Apple Inc. Keychain Services. Available online: https://developer.apple.com/documentation/security/keychain_services (accessed on 19 April 2022).
29. Google LLC. Firebase Helps You Build and Run Successful Apps. Available online: https://firebase.google.com (accessed on 24 April 2022).
30. Google LLC. Upload File Data. Available online: https://developers.google.com/drive/api/guides/manage-uploads (accessed on 10 April 2022).
31. Apple Inc. DownloadTask. Available online: https://developer.apple.com/documentation/foundation/urlsession/1411511-downloadtask, (accessed on 25 April 2022).
32. ITU-T. Overview of the Internet of things. In *Recommendation Y.4000/Y.2060*; International Telecommunication Union: Geneva, Switzerland, 2012.
33. Balogh, S.; Gallo, O.; Ploszek, R.; Špaček, P.; Zajac, P. IoT Security Challenges: Cloud and Blockchain, Postquantum Cryptography, and Evolutionary Techniques. *Electronics* **2021**, *10*, 2647. [CrossRef]
34. Kluczniak, K. Witness Encryption from Garbled Circuit and Multikey Fully Homomorphic Encryption Techniques. Cryptology ePrint Archive, Paper 2020/1502. 2020. Available online: https://eprint.iacr.org/2020/1502 (accessed on 22 August 2022).