International Journal of Mathematics, Statistics, and Computer Science
Vol.2, 2024
ISSN: 2704-1077 eISSN 2704-1069, DOI: doi.org/10.59543/ijmscs.v2i.7971

Comparing Several Encrypted Cloud Storage Platforms

Chinnadurai Manthiramoorthy[1,*], K. Mohamed Sayeed Khan[2], Noorul Ameen A[3]
[1]Software Architect, Abbott Laboratories, Burlington, USA; chinna@bu.edu
[2]PG & Research Department of Computer Science, Sadakathullah Appa College, India; mrkhan1031@gmail.com
[3]Tiger Analytics, Chennai, India; mailtonoorul@yahoo.com

Received 10.06.2023, Revised 21.07.2023, Accepted 01.08.2023, Published 06.08.2023

ABSTRACT: Cloud services and cryptographic cloud storage systems have gained popularity in recent years due to their availability and accessibility. The present systems are nonetheless still ineffectual. They are the best since they demand a lot of trust from the user or the provider. To ensure they are not violating any End-User License Agreement (EULA) clauses, providers typically keep the ability to examine the files that have been saved, and some even keep the ability to share the data. It is simple to create a copy of every piece of data when a provider has access to go through it, which is considered an abuse. A typical user would have a very difficult time proving these claims because they have no method of finding any evidence supporting such claims. Due to the growing quantity of Machine Learning (ML) performed on personal user data for either tailoring advertisements or, in more severe cases, manipulating public opinion, this issue has only gotten worse in modern times. Due to the volume of users and files kept, cloud storage services are the ideal location for getting such information, whether personal or not. To retain complete anonymity, the user could take the simple step of adding a local layer of encryption. This will prevent the cloud provider from being able to decrypt the data. The requirement for ongoing key management, which becomes more challenging as the number of keys rises, is another drawback of this. To better understand normal behaviours and pinpoint potential weaknesses, this study aims to explore and assess the security of a few well-known existing cryptographic cloud storage options. Among the vendors investigated are Microsoft Azure, Tresorit, Amazon S3, and Google Cloud. Based on documentation particular to each service, this comparison was made. However, most providers frequently provide only a limited amount of information or don't go into detail about specific ideas or procedures (for instance, security in Google Cloud), leaving room for interpretation. The authors conclude by outlining a unique approach for encrypted cloud storage that employs Cocks Identity Based Encryption (IBE) and Advanced Encryption Standard (AES)-256 Cipher Block Chaining (CBC) to limit potential abuse by alerting the user anytime a file inspection takes place. Cocks IBE will be utilised as an alternate cryptographic method for access controls, and AES-256 will be used for the Initialization Vector (IV) features' encryption. Additionally, Fiat-Shamir authentication will be zero-knowledge. A system like this might be used by companies who offer services in the actual world because it would boost customer confidence.

Keywords: Stroage systems, cloud computing, Google, Amazon S3, Cloud services, Cryptographic, Google Cloud, Microsoft Azure, Tresorit.

1. INTRODUCTION

Cloud storage offers high data availability, easy access to data, and lower infrastructure costs by storing data with distant third-party providers. However, users regularly need guarantees about the integrity and confidentiality of specific types of data, which contemporary cloud storage solutions are unable to provide without incurring astronomical costs in computation and bandwidth. As a result, availability is frequently insufficient. Integrity and confidentiality are essential for high-impact business data, top-secret government documents, and medical information, to name a few examples. Cloud computing services are in greater demand. Therefore, it is more challenging to handle, store, and assess data given its increasing amount. Users who use cloud storage services, such as Amazon S3 and Microsoft Azure, gain access to advantages like elasticity, accessibility, and dependability. All of these services, however, face security and privacy issues, which has spurred research on secure cloud storage systems [1]–[5]. However, the typical approach is to encrypt and sign data at the user using digital signatures and symmetric encryption to achieve integrity and confidentiality features in storage systems (whether cloud-based or not). This kind of storage, also known as cryptographic file systems, offers end-to-end security in that data as soon as it is released from the user's control. While cryptographic file systems provide great security guarantees, their functionality is severely constrained, rendering them inadequate for current storage systems. The practicality of cryptographic file systems is constrained by two important drawbacks. The first drawback is that file system traversal has mostly been replaced by the search for retrieving data. The need for search in storage systems has long been understood, which has led to significant research into semantic file systems that completely rely on search-

based access rather than the typical hierarchical file/folder structure [6]–[9]. Programs that arrange knowledge for a user to facilitate speedy searches have also been developed because of the increased relevance of discovery. Since searching is presently the most practical way to get information, search programmes like Apple Spotlight, Google Desktop, and Windows Desktop Search have quickly become essential. The second drawback is that cryptographic file systems only provide data retrieval-integrity guarantees. As a result, this study offers a mechanism to examine the security of a few popular current cryptographic cloud storage alternatives, including Tresorit, Google Cloud, Amazon S3, and Microsoft Azure. The authors also create a novel method for encrypted cloud storage that makes use of Cocks IBE[1] and AES-256 CBC[2] to prevent potential abuse by notifying the user whenever a file inspection occurs.

The following is the paper, the studies pertaining to cryptographic file systems will be covered in the next section. A comparison between several cloud service providers is made in section III. A general comparison is provided in section IV. The paper's prospective storage solutions are proposed in Section V, and findings and suggestions for more study are presented in Section VI.

2. LITERATURE REVIEW

End-to-end security is provided by cryptographic file systems, guaranteeing integrity and confidentiality even when using an unauthorised storage provider. Numerous studies on cryptographic files have been conducted. For example, Plutus [10] is a cryptographic file system that permits per-file sharing but lacks secure key revocation [11]. An encrypted cloud storage system called SiRiUS [1] offers per-file sharing and a weakened version of freshness, which prevents a file's metadata from being altered with an earlier version of the same file. A read/write shared cryptographic cloud storage system is CloudProof [12]. It achieves vitality, fork consistency [13], integrity and confidentiality. Fork consistency makes it possible for an untrusted storage service to offer various copies of the same data to multiple users without being noticed. Participants in the system can show a third party that any of these features have been broken in addition to these features by using CloudProof. Applying symmetric encryption, digital signatures, or message authentication codes (MACs) to user data implementing well-known cryptographic algorithms is a typical strategy for ensuring end-to-end security. The problem of discovering symmetrically encrypted data may be completely resolved by unaware RAMs [14]. Sadly, these methods lack effectiveness, which is logical given that they give off a strong sense of security. In [15], they explicitly investigated searchable encryption. They provide a non-interactive method that offers search time that is primarily linear in terms of the size of the file collection. In another study [16], they offered a Bloom filter-based design that results in false positives and requires an $O(n)$ search time on the server (where n is the number of files). Unfortunately, the latter is impracticable since the asymptotic search time has huge constants. A similar strategy was proposed by Chase and Kamara [17], however, it has a low level of space complexity. In the public-key setting, searchable encryption has been studied by Boneh et al. [18]. In addition to these studies, interactive protocols were used in the storage documents, allowing a client to verify the accuracy of data stored remotely such as Proof of Data Possession (PDP) [19] and Proof of Retrievability (PoR) [20]. A PDP generally assures that tampering will be discovered if it rises above a certain threshold. On the other hand, if the tampering is below a predetermined level, a PoR can provide additional assurance that the data can be recovered. PDPs and PoRs were both given expansions and improvements by [21]–[25].

3. COMPARISON

In this section, a thorough comparison of Tresorit, Google Cloud, Microsoft Azure, and Amazon S3 is provided.

3.1. Tresorit

Beginning with an explanation of a few specific Tresorit components in the following sub-sections.

3.1.1Tree-based Group Diffie-Hellman (TGDH)

There are benefits and drawbacks to distributing keys through a single Central Authority (CA). Every group member must contribute equally to the cost of using the contributing group key management system known as Tree-based Group Diffie-Hellman (TGDH) [26]. Each message is digitally signed, and the communication channels are declared secure. The public blinded keys of the other child nodes are combined with the key of

[1] Clifford Cocks proposed the Cocks IBE scheme, an IBE method, in 2001. The hardness of the quadratic residual problem serves as the foundation for the scheme's security.

[2] A block of bits, or a unit of bits, is encrypted using a block-wide cypher key in this type of block cypher operation.

one of the child nodes to create a non-leaf node's key is located. It stresses that the graph's root should not be utilised as an encryption key. Instead, it is recommended first to employ a strong hash function[3] and then utilise the output as an encryption key. In TGDH, five operations are accessible:

- The group gains a new member (Join).
- One of the group's members has left (Leave).
- One group and another were combined (Merge).
- The group's splits (Partition).
- Making a fresh group key (Key refresh).

In addition to the operations, the following cryptographic characteristics are guaranteed by TGDH:

- Key confidentiality in a group prevents a passive attacker from obtaining the keys using brute force.
- Upfront confidentiality means that no new keys should be known to a user who leaves the block.
- Reverse confidentiality, which states that a person entering a group must be unaware of the previous keys.

Keyword isolation prevents an attacker from ever using his information to obtain further keys after using a subset of the keys.

3.1.2Invitation-Oriented TGDH (I-TGDH)

Due to the rigorous selection criteria that must be met by the sponsor, the initial version of TGDH is not appropriate for scenarios involving public sharing [27]. There should be a method for determining whether a user has the rights required to join the team when they send a join request message. Finding out if there is a member in the entire tree who welcomed the new user will quickly fix this. This tactic, however, may result in delays because the inviter and the sponsor may never be able to use the application. This is especially true because the sponsor's allocation is highly dependent on the tree structure. It would be great if a user could join the group right away after receiving an invitation. The authors of [27] propose Invitation-oriented TGDH (I-TGDH), a variant of the original that is more suitable for cloud-sharing scenarios and keeps its efficiency and cryptographic properties. The idea of a shadow node is presented by the authors. Only leaf nodes that hold temporary private keys can act as sponsor nodes to allow tree rebuilding without relying on a particular node. For instance, in Fig. 1, for *m3* to add *m4*, *m3* must first recreate the previous tree using both *m4*'s blind key and their private key. Since *m3* is the sponsor node, this happens. The recalibration cannot take place if *m3* is not the sponsor node, or the sponsor node must connect to *m3* to rebuild the tree or exchange its private key. The computation can't be done by another sponsor node because *m3*'s private key shouldn't be made public, thus the new sponsor node must wait for *m3*'s response. Without having to wait for *m3*, the sponsor may now recreate the tree if they are utilising a shadow node.

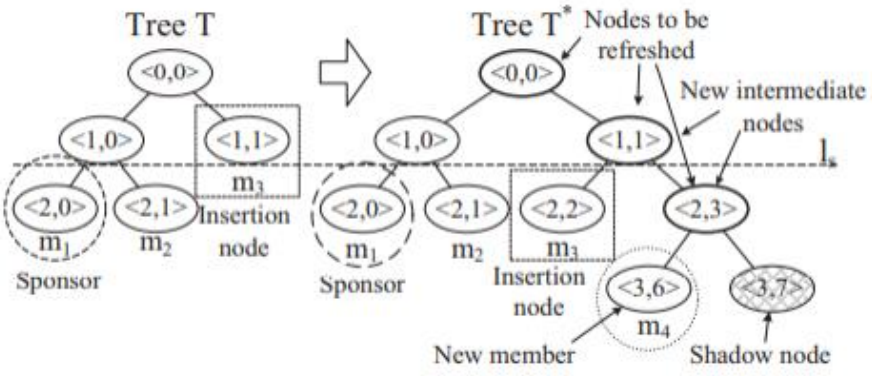

Fig.1 Using shadow nodes to insert nodes.

[3] Regardless of the input, a hash function converts any dataset into a fixed-length character sequence.

3.1.3Secure-TGDH (S-TGDH)

A modified version of TGDH called Secure-TGDH (S-TGDH) was created for group administration on ad hoc networks. The authors constructed an authentication scheme in [28] on top of several upgrades. The Asokan Ginzboorg[4] protocol, another method modified for use in the authenticity algorithm, is as follows:

1. $M_n \rightarrow M_i$ *: {public$_n$, c1i}$_p$, n1, signature$_n$ (i=1…….n-1)*
2. $M_i \rightarrow M_n$ *: {public$_i$ c1i, c2i}$_{public_n}$, n2, signature$_i$ (i=1…….n-1)*
3. $M_n \rightarrow M_i$ *: {c2i}$_{public_i}$, n3, signature$_n$ (i=1…….n-1)*
4. $M_i \rightarrow M_n$ *: agree ,n3, signature$_i$ (i=1…….n-1)*

Where, *n1*, *n2*, and *n3* are all nonces[5]; *C1i* authenticates the node's identity *Mi* and *C2i* also, *Mi* can authenticate *Mn*'s identification. *{x}$_p$* denotes x is encrypted using key p, where p is a shared weak secret among all users of TGDH tree share. To prevent a scenario in which every node sends a message to every other node, which could result in delay and other issues, an initiator node (*Mn* in this example) will first be chosen. M_n uses encryption employing *p*. M_n can validate the signature and decipher the challenge after obtaining the message, to safeguard his public key, a challenge, a nonce, and a signature. Then it will create its challenge, create an encryption key using the first challenge and *Mn*'s public key, raise the nonce, sign the message, and send it back. The decryption outcome ought to match the *Mn* challenge if the decryption was carried out effectively and *Mi* was aware of the secret *p*. This is how *Mn* checks to see if *Mi* is real. For this type of authentication, a shared password must be used to create the group (the weak secret *p*). This version retains all the original attributes of TGDH.

3.1.4Peer-to-Peer (P2P) Group Data Sharing in Tresorit

When the system is Peer-to-Peer (P2P), it should be highlighted that all modules will exist on the user end; a primary server is not implied. This approach is also used by Tresorit with some slight adjustments. For instance, there can only be paying users since it does not provide service to non-paying users continuously. The cloud provides all of the resources; thus, users do not have to contribute any of their own, which suggests that the Cloud Communication Module (CCM) and CPA modules must be modified. Since it maintains keys, the placement and implementation of the Authenticating and Agreement module (AAM) module are equally crucial. The provider might have simple reading access regardless of where this component is placed on the server. Additionally, it is quite easy to instal a backdoor on the user's computer that automatically delivers the server's login credentials whenever they are needed. Package research or heavily utilising the user would be needed to refute the latter. Therefore, a File Level Module (FLM) is used by Tresorit to control read and write permissions. Such a key is used to decrypt files that have been encrypted using the CBC mode of AES with a 256-bit key size. The FLM module produces these keys using a hierarchical key lock box architecture, which reduces the need to change the master key when a particular file key is changed. To verify write rights, an RSA-digital signature is used. But another algorithm might be used in place of both of those such as lazy re-encryption. The exchange of information happens across a channel that has been authenticated or that is based on an authenticated key exchange protocol. Not all group members need to be able to utilise the computer when a new member joins or leaves. The authentication process and group modification management are handled by the AAM. For authentication, it uses S-TGDH or RSA-based authentication [29]. To view the data item shared with that group, a member of the group must first declare the ID of that group. The process of group-specific authentication will then start. The user gives authentication information, such as a password, an RSA private key, or other credentials, using an auxiliary module. After confirming this information, the AAM module, depending on the accuracy of the input, will provide the read parent secret key or the written master's secret key. The information is then used to create signatures or decode keys that are unique to files. The keys and signatures connected to each file will likewise be tracked by the FLM [29].

As was already mentioned, Gradual re-encryption is used by Tresorit, although it has a significant flaw. Until the key associated with that file is changed, removed users can alter the contents of files. Even if all the write keys were immediately changed, the creators of Tresorit understood that the deleted individual would still have a great opening of opportunity to engage in illegal activity. It is highly advocated only granting write permissions to reliable users as a result [29]. In the real world, unless the attacker had a specific goal in mind, their edge would rely mostly in their ability to quickly alter their course of action after being expelled from the cluster. However, a record that includes a time stamp and the username of the user who edited or uploaded the

[4] A protocol for creating a group key in a Bluetooth ad hoc network.
[5] An arbitrary number that can only be used once in a cryptographic transmission is known as a nonce.

content should be retained. Legitimate group members will therefore be able to identify which files were edited and when they were modified if an attacker leaves the group and immediately makes a modification. Knowing another user's key would be necessary to fake such records. The installation of malware that launches automatically when downloaded is the most dangerous scenario here. The hostile user can easily act this way while still a member of the group, thus he does not need to be kicked out. Toggling all of the write keys at once could, depending on which user performs the computations, cause Denial of Service (DoS) if the tree structure is large. In conclusion, this method would be helpful in extremely extreme edge-case circumstances, but it would also pose more serious security concerns, which is why it is not used.

3.1.5Tresorium

Tresorit's cryptographic data storage system is called Tresorium. Most cloud providers maintain some amount of control over their users' data while it is stored on their servers, which is frequently a positive sign that indicates a privacy violation. Each instance of this offence will decide its severity. While most cloud service providers keep the option to inspect data stored on servers only to verify that the EULA is being followed, some choose to use it for online marketing and, in rare circumstances, cooperation. Here, we have the same key structure as in the previous section. Its structure would match that of the provided directory where the keys are stored. The master lock box will contain the group key, keys for decrypting the root directory's data, and keys for decrypting the root directory's components, allowing access to the lock boxes below inside the tree-like framework. The I- TGDH technique is used to determine the group key. The cloud can access the keys and, consequently, the encrypted data if the AAM is only present on the user's computers. This is so that the AAM may provide the lock box's keys, which contain the decryption key.

3.1.6 Workflow in Tresorit

Trezor creates a cryptographic random salt of 160 bits of random data as soon as a user registers. This is done to ensure that the database's information contains the fewest possible specifics about the password itself. The Password-Based Key Derivation Function-2 (PBKDF2)- Hash-Based Message Authentication Code (HMAC)-Secure Hash Algorithm-1 (SHA)[6] algorithm is applied to the input, the password is concatenated with the random bits, and the function is once more run on the output. This procedure is carried out 10,000 times locally by the user. The database is employed to record both the 160 random-bit string and the complete version, which are then implemented in the zero-knowledge authentication procedure. The following describes how the zero-knowledge authentication procedure functions [30]:

1. The operator initially informs the server of their desire to log in.
2. The 160-bit random-bit string used during registration will be sent along with a task request from the server.
3. The same as when registering, the user uses the PBKDF2 function, and the server receives the outcome.
4. The server makes a comparison between the incoming value and the database value. The user has signed in successfully if the response is yes.

After the user has logged in, a secure Transport Layer Security (TLS) channel needs to be established before data transmission can start. Hardened TLS, a TLS variation used by Tresorit, enables TLS authentication for both users and servers. As a result of the end-to-end encryption provided, the authors say that this is done to prevent attackers from impersonating them and accessing the data on the cloud [30]. Only the user possesses the necessary keys for performing such actions, enabling them to select the encryption key to be used for a specific file in Tresorit. Encryption and decryption operations are solely carried out by the user within the system. The AES-256 cryptosystem is employed in CFB mode. As mentioned before, this mode of operation necessitates the use of an IV, which, if mishandled, could result in data breaches. By establishing a new IV for each encrypted object, this is avoided. Before the message is delivered to the cloud, a message authentication code (MAC) is produced for the file that has been encrypted (HMAC-SHA512 in this example) and appended to it to ensure its authenticity. The data are then obtained by the cloud and stored there while maintaining their original structure. The lock box layout must first be modified since the architecture described in the above

[6] PBKDF2 is a popular technique for generating keys of a specific length using inputs such a password, salt, and the number of repetitions. In this instance, it specifically employs the RFC2898-recommended SHA-1 hash function using HMAC.

section is used here. The module will also contain a pre-master secret and user-specific certificates if the AAM is RSA-based. The key recovery process is the same as what was previously mentioned such as:

1. Specific information is sent to the AAM to discern the master's secret.
2. Any file-specific keys would then be decrypted using the master's secret so that the contents could be accessed.

For performance reasons, the actual stored data is encrypted with AES-256, while the lock boxes use asymmetric cryptography to get rid of key management issues. Sending an email to welcome a new user to a community will start a three-step authentication process:

1. Using their RSA keys and an X.509[7] certificate, users will be able to identify one another. We are also creating a permanent symmetric key here, which may be used to invites in the future. A private 256-bit code is further provided by the individual sending the invitation.
2. By answering a question with the trick learned in step 1 to a challenge, the invited user must now prove that he was invited. He joins the group if his response is accepted.
3. Users can use a pre-shared password for increased security (Optional).

If a user is expelled from a group, his access to and ability to change files may also be suspended. This can occur in a typical setting for a variety of reasons, some of which are slightly more benign than others. Tresorit also uses this method in three phases:

1. The user's name is taken off the group's Access Control List (ACL). This is a common practice in most encrypted cloud storage services that support group sharing and management. Typically, this step alone would suffice if users had confidence in the cloud service.
2. The certificate belonging to the user who was removed is erased from the Authentication and Authorization Module (AAM). Additionally, the person who removed the certificate modifies the keys in the main lock box and then re-encrypts it. As a result, the removed user is denied access to the new root directory. However, they can still read previously accessed files by utilizing the cache memory.
3. All files that were accessed by the deleted user now have the "dirty flag" assigned to them. This flag allows for gradual re-encryption of these files with a new key as they are modified by users who have write access.

3.2.Google Cloud

Large and well-known commercial cloud service provider Google Cloud also offers Platform as a Service (PaaS), Software as a Service (SaaS), and Infrastructure as a Service (IaaS) in addition to storage. The four tiers of security are infrastructure concerns, encryption at rest, encryption in transit, and application layer security which are discussed in detail in the following subsections.

3.2.1 Infrastructure Security and Other Aspects

Cryptographic signatures are utilized by Google Cloud to verify the integrity of computers and the software they run, ensuring that no unauthorized tampering has occurred, in addition to securing the physical premises by controlling access. Every physical computer may be individually recognised, and every signature is based on a microcontroller, or a security chip created by Google. The infrastructure is controlled by a system known as "Borg," which has no faith in any of the facilities employing it. Before any communications can take place, all services must first be cryptographically verified and authorised implementing a set of credentials. A higher level of service separation and sandboxing is needed because Google Cloud offers a variety of cloud services on top of secured storage. The following techniques are employed by Google Cloud to overcome this problem: Hardware virtualization, language- and kernel-based sandboxing, and Linux user isolation. The suitable methods and amount of isolation to be applied depend on the level of security necessary for a certain operation or service. It is crucial to keep in mind that hardware virtualization may be adequate for cryptographic cloud storage services if the supplier makes sure that no stored file may be executed. Contrarily, Tresorit solely encrypts and decrypts data on the user's workstation, making it unlikely to run files straight from the cloud

[7] An X. 509 certificate securely links websites, people, or organisations to cryptographic key pairs of public and private keys.

storage as a matter of normal procedure. However, a user may provide a plain text file that is mistakenly recognised as encrypted and then starts running immediately after being saved. To govern access between services, structures resembling ACL might be utilised. Each service and user is assigned a special identifier, which decides whether or not they are allowed to use particular services. Before transmission, all communications between services are encrypted. It is possible to change the security level, from straightforward verification of integrity to more powerful double-layered encryption. To further improve isolation, encryption is also used at the application layer. A key management server that enables automatic key rotations centrally manages the keys used for this encryption. The implementation of sophisticated filtering techniques and explicit DoS security is possible because only a piece of the infrastructure is directly connected to the internet [31]. This functionality is not explicitly supported by Tresorit.

Any cloud service must guarantee high availability, but PaaS, SaaS, and IaaS-based cloud products are especially dependent on it because it might be difficult to quickly replace these services. Local data storage, on the other hand, can be used as a workaround when a storage service is down. In its zero-knowledge execution, Tresorit takes a comparatively straightforward approach to authentication by asking users for their username and password. Google also keeps track of the devices and places where users have signed in, documenting the IP address or sign-in location in the process. There are several ways to check whether a login was successful on a certain device, such as keeping information that may be used to identify the terminal specifically, including MAC address storage (which is simple to spoof), or storing information on the user's device (which can be considered intrusive). The privacy of a user is compromised by location storing and device fingerprinting. By using a public key architecture, Tresorit addresses this issue without retaining any location-related information. There will be numerous X.509v3 certificates used to authenticate each device. The credential's storage method and hashing algorithm are unknown. It has been stored using the conventional method if it is utilised at all. Furthermore, the provider already learns a great deal about the users because authentication is not zero-knowledge. The use of two-factor authentication is required. They both offer it. Due to the amount of data stored, extensive security measures are taken to protect it [31], including requiring personnel to utilise phishing-resistant One Time Pads, actively monitoring infrastructure admins, and two-party authorization. By combining intelligent robots with a body of rules, intrusions can be quickly recognised. Additionally, available around the clock is the threat rescue team, adding still another level of dependability. To address this challenge, Tresorit maximizes the amount of stored data and minimizes its reliance on external sources. In the event of a data breach, potential attackers would only gain access to encrypted or uninteresting user information, such as hashed data or saved encrypted data, and the only information that would be collected that was useful would be the user's email address. The most dangerous case is a serious system breach when the attacker has access to view and modify application code. Due to its temper-proof computers and software, Google Cloud first seems to have the advantage because a breach would be quickly discovered. On the other hand, Tresorit benefits from the Microsoft Azure system.

3.2.2 Encryption at Rest in Google Cloud

All client data and information is encrypted by Google Cloud using AES-256 or AES-128 and the CBC, Galois/Counter Mode (GCM), or Counter Mode (CTR) of operation, along with a proof of integrity. The application layer, platform layer, infrastructure layer, and hardware layer make up the four levels that make up the Google Cloud data platform. Except for the application layer, all levels of the data are encrypted and separated into sections. For identifying purposes, each component is given a special tag and key, and before being stored, it is scattered over many places and encrypted. Furthermore, collisions are quite infrequent. Users cannot require automatic re-encryption for every file, which is like the idea of lazy re-encryption. An ACL that identifies who has access to each chunk is also present. Tresorit uses the directory tree structure for general storage and the TGDH structure for key sharing. Because the key can only be rebuilt by the members of the tree, TGDH sacrifices confidentiality for speed. If the data chunks can be accessed fast, the data tree structure can be created more quickly. If a key is stolen, Tresorit should be at a disadvantage. The level of the directory tree at which the compromised key was in Tresorit significantly affects the damage because every child node would be impacted and put in danger. As a result, if the master key is found, the entire tree is destroyed. The recovery process may take a while because of the slow re-encryption. However, targeted malware is the most likely way for a key to be compromised, hence it is the user's responsibility to manage keys. The ACL in Google Drive can function in this circumstance as an additional layer of security, but depending on how it is configured, the provider may acquire higher permissions and demand more trust from the user. If the data is saved on a hard drive, the hardware layer encryption is AES-128, and if the data is saved on a Solid-State Drive (SSD), the hardware layer encryption is AES-256. Additionally, some information is stored in plain text,

including temporary files, core dumps, debugging logs, and console logs [31]. This information has no impact on encrypted storage. Due to the absence of file execution, they are unable to generate these logs on their own. This might be a privacy issue in the case of SaaS and PaaS since they would require a manual inspection to fix the problem.

Google maintains a custom library called BoringSSL. The decision was taken to stop using the OpenSSL library because of its multiple vulnerabilities in security. The data encryption keys, and key encryption keys are now both handled by Google Cloud's key-handling service. BoringSSL was used to generate these keys. Data pieces are encrypted and decrypted using data encryption keys; for better productivity, these operations are kept near the relevant data blocks. Employing the key-encryption key, the data encryption keys are further encrypted. Either the management service or the storage service can produce the keys for key encryption. The usage of these keys is restricted by an ACL, they are kept in a key management service, and thorough records of who used them and which key was unlocked are kept. To optimize key management processes, the data encryption keys can be reused, leading to improved efficiency. Unlike Tresorit, the data encryption keys in Google Cloud are not kept private. Google Cloud possesses all the necessary information, including the location of the encrypted data (or the capacity for intercepting it) and the key to decode it decrypt and access the stored data. This holds true even when the key transfer occurs within a local network, making external interception virtually impossible. Utilizing asymmetric cryptography for key transfer between the management solution and the user is a straightforward approach, but it proves to be ineffective in preventing easy interception. Alternatively, the user can keep the information locally rather than provide it for storage. The key management service provides a distinct data encryption key and a random seed that is used to encrypt and restore files separately from the original data. The data can only ever be viewed in its encrypted form as a result. Since it would be impractical for the host to encrypt the data in the first place, Tresorit does this by default. They have their exclusive storage area called the root key management service, and they are AES-256 keys (AES-128 for earlier keys). Another key, the root effective management service master key, which is also AES-256 and only exists in RAM, protects those keys. It could take a while to complete this process from top to bottom for larger files. Since each chunk needs its unique key, the first stage takes a while; however, if a single all data encryption keys are encrypted using the same key-encryption key, and the encryption algorithm uses a process becomes more streamlined, the second stage might be shortened. Since moving between levels requires unlocking lock boxes and decrypting files, the depth of a particular file in the tree structure is the main factor slowing down decryption in Tresorit. The number of users with whom the file was shared will be known if it is shared. The one drawback of TGDH is that if the file was shared with many individuals, recreating the key can take longer.

3.2.3 Encryption in Transit in Google Cloud

The security procedures for processing local traffic are different from those for dealing with external traffic due to the sheer size of Google Cloud and for enhanced security. Google Front End (GFE) is a dedicated module that only manages external traffic to support this strategy. After the user, the certificate has been validated and Hypertext Transfer Protocol Secure (HTTPS) connections using TLS have been created, this module responds. All information sent or received using Google Cloud services is encrypted, authenticated, and includes an additional integrity check on top of TLS. This is the same as Tresorit because it uses the mutual authentication-supporting Hardened TLS version. There, data integrity is ensured by using HMAC-SHA-512. Additionally, Google has a TLS implementation that ensures forward confidentiality, according to [31]. Forward confidentiality is an integral part of the TLS 1.3 protocol, rendering certain practices less relevant. However, it's important to note that BoringSSL does not currently provide complete support for TLS 1.3 as of March 2018. Additionally, Google has been operating its own Certificate Authority (CA) since June 2017, allowing them to issue certificates for Public Key Infrastructure (PKI). However, for broader compatibility, Google continues to rely on third-party CAs as their CA must be trusted by various operating systems and browsers. To reduce the likelihood of them being compromised, CAs are very sometimes brought online in practice. As a result, some groups might find it useful to be able to sign such certificates whenever they like rather than having to wait for another CA to establish itself. Instead of going through the entire process again, entities that have successfully executed a TLS session before can connect using a private session ticket or ID. Google consistently updates TLS certificates to greatly minimize the risk of a TLS session being compromised by an attacker. For secure data transmission from one VM inside the private network to another VM located outside of the Google Cloud area, AES128-GCM encryption is employed at the network layer. Each session's unique encryption key is produced using Application Layer Transport Security (ALTS). Authentication is performed via a security token. This token contains a physical boundary secret as well as data that specifically identifies the VM. Communication between services has been encrypted, validated, and guaranteed to be legitimate using ALTS. Users of Google Cloud can change the methods utilised for in-transit encryption. There

are several solutions, ranging from using TLS/ Secure Sockets Layer (SSL) for more intricate TLS/SSL control to using Google Cloud Virtual Private Network (VPN). Because the user may always encrypt the same material locally using third-party software and save the key locally, this functionality isn't really helpful for data storage. All storage providers give some resistance against quantum computing in terms of post-quantum cryptography. In addition to allowing RSA-2048 authentication, which is susceptible to TLS and ALTS employ elliptic curve-based key transfer techniques, the elliptic curve DSA providing authorization, and elliptic curve cryptography to protect against quantum computing for AES-256 for encryption. While Google Cloud still utilises AES-128 for some data, Tresorit exclusively uses AES-256 for storage.

3.3 Microsoft Azure

Another well-known cloud service provider created by Microsoft is Microsoft Azure. Similar to Google Cloud, it provides PaaS, SaaS and IaaS. Microsoft Azure data centres are also used by Tresorit.

3.3.1 Infrastructure Security and Other Aspects

The physical security of the data centres is the initial security precaution implemented, comparable to Google Cloud. Similar to deletion, everything that cannot be removed is physically destroyed on the premises during data wiping. To prepare for potential natural disasters, data is backed up in two separate locations with three copies kept in each [32]. A service that maintains VMs is called Fabric Controller. To ascertain whether a VM has been compromised at starting, a secure boot procedure is used. To validate all of the components before loading, this calls for the use of digital signatures and two databases [32]. Cerberus, a chip whose identity cannot be duplicated, is the foundation of everything. Unlike Google Cloud's microcontroller/security chip, which omits this detail, this makes it impossible to copy the chip. Users can also encrypt virtual hard drives, which, unless a key controlling authority is also compromised, can be seen as an additional line of defence in the event of security. The manner in which Microsoft Azure controls inter-service communication is another similarity to Google Cloud. Identification and communication between elements are established using TLS encryption and X.509 self-signed certificates. Microsoft's CA is supported by a reliable root CA in contrast to Google Cloud's CA. In other words, the certificates are not limited to usage in cloud environments. To authenticate itself to other services, the Fabric Controller purchases such a certificate and uses it together with a set of keys or passwords. An edge network, a wide area network, a local area network, a regional gateway, and a central network are the four components that make up the network infrastructure. To provide access control at the network layer, another firewall is employed. The justification for this architecture is the expansion of regions and data centres. To ensure robust security measures, Google employs multiple layers of firewalls to filter communications and prevent unauthorized access. Additionally, forced tunnelling is implemented to enhance the security of Virtual Machines (VMs). Forced tunnelling restricts VMs from initiating requests outside the local network but allows them to respond to incoming requests. This approach effectively mitigates the risk of malicious users tricking VMs into sending requests and receiving malware in response. Microsoft Azure offers additional monitoring tools in addition to doing a vulnerability scan four times a year. Every build and component is scanned for viruses using Microsoft Anti-Malware before release. Prior to release, each build and component undergoes comprehensive virus scanning implementing Microsoft Anti-Malware. An incentive program is in place to reward users who identify vulnerabilities, particularly those related to remote code execution, in the Azure hypervisor hardware virtualization tool. Various techniques including VPNs, TLS 1.3 and newer versions, Internet Protocol Security (IPsec), among others, can be utilized to ensure the security of data during transit. However, it's worth noting that enabling TLS 1.3 can introduce the risk of downgrade attacks, where attackers attempt to exploit weaker cryptographic primitives available in TLS 1.3, potentially compromising the integrity of the data. This depends on the implementation, though. TLS 1.3, as previously mentioned, does not completely provide forward confidentiality even without this. As of March 2021, Microsoft OneDrive is still using TLS 1.3, making it unsafe.

3.3.2 Encryption at Rest in Microsoft Azure

The client or server side of encryption can be chosen by the user. Key management is the responsibility of the user if encryption is done on the client. The best encryption technique for the actions taken can also be chosen by the user. Given that Tresorit uses Microsoft Azure data centres, their plan is perfectly conceivable. As a result, none of the keys is accessible by the cloud. Because the server handles keys and decides on an encryption method when utilising server-side encryption, the client forfeits privacy for convenience. The TLS protocol is

the only one used by default when using server-side encryption, therefore if the TLS channel is ever compromised, the attacker may view whatever data was being sent now in plaintext. With client encryption, on the other hand, the attacker can only view the encrypted data. Tresorit's client-side encryption method by design offers convenience and privacy. Only server-side encryption is offered by Google Cloud. A hybrid approach where the user is responsible for key management while the server handles the encryption is feasible, but it poses security vulnerabilities during data transmission. A user will additionally lose possession of the data encrypted with that key if it is lost or misplaced. The ability to refresh the keys whenever the user wants is the only benefit of this strategy. The keys in Tresorit are updated whenever a group is modified. However, the file that needs to be re-encrypted must first be modified before the re-encryption can occur. This can be a concern for companies that maintain many files that are never updated because re-encryption might not take place in certain situations. Like Google Cloud, Azure Blob storage also implements server-side encryption at rest. The data is separated into pieces, each of which is encrypted using a different data encryption key, which is then encrypted using an encryption key. This asymmetrical group is also responsible for managing VM encryption keys. The cloud also stores the encryption keys for the encrypted data. Users can also use a secret private key to access their data. This security method falls between Google Cloud and Tresorit in terms of the level of difficulty involved in obtaining plaintext data in the event of a security breach. This is mainly because there are fewer steps involved in getting a key than there are with Google Cloud. To acquire plaintext data, an attacker must undergo several steps. Firstly, they need to locate the data chunks and identify the corresponding key chunks used for data encryption. Furthermore, it is crucial to both acquire the required key-encryption keys and decode the data. Additional protection against assault is provided by Google Cloud's root key administration service and root key owner key distributor. Since both services use ACLs, impersonating a member of the ACL or the data owner is one possible method to trick a key administration entity into decoding a keys for an attacker. To protect employees from phishing and impersonation efforts, Google Cloud grants them higher levels of access. However, such impersonation attempts have minimal impact on Tresorit, because of its secure transmission and lack of cloud-based key storage/management. It is important to note that the number of decryption steps involved may impact data access speed based on the AES mode of operation. For example, Galois/Counter Mode (GCM) allows for parallelization, leading to faster decryption and encryption. In Microsoft Azure's Azure Data Lake, encryption is performed using three keys: the key for encrypting the master key, the key for encrypting blocks, and the key for encrypting data. However, the block key used for encryption is produced merely from the data blocks and the information's encryption key, and it's never maintained, whereas the primary encryption keys are rotated only when the user requests it. An additional layer of security is added by encrypting both the data encryption key and the master encryption key. Azure Data Lake enables data analysis in addition to storage. Identity management is therefore a crucial element. In Microsoft Azure, this can be done in several ways, including inter-authentication, back proxy, single sign-on, and azure Role-Based Access Control (RBAC) [32].

Single Single sign-on (SSO) enable users to log in once, typically involving the retention of specific cookies on their browser. However, this approach can introduce vulnerabilities such as merge scripting and cross-site response forgery attacks, depending on the browser being used. Tresorit is also susceptible to these attacks, but their impact on the system depends on factors like the contents of the cookie, how the certificate is stored on the user's computer, its connection to the cookie, and whether sensitive data is stored within it. If an attacker manages to obtain all three pieces of information, they can log in and assume the user's identity. However, with just two pieces of information, the attacker cannot impersonate the user without the necessary certificate, nor can they decrypt the data without the required key. Two-factor authentication provides enhanced security as the attacker would need access to the authenticator in addition to the password. Typically, the authenticator generates a time-limited One-Time Pad. In Microsoft Azure, SSO is implemented differently using a user-owned private key. Depending on how the Trusted Platform Module (TPM) functions, the user may become a target of targeted malware attacks aimed at stealing a secret code or key. In its present configuration, the protocol is also very reliant on the additional TLP that is utilized on top of it, only the user uses an asymmetric key to confirm his identity to the authenticator, making it vulnerable to man-in-the-middle attacks. An anti-phishing Universal Serial Bus (USB) device called a Fast IDentity Online-2 (FIDO2)[8] security key can also be used by users to verify their identity. In Google Cloud, phishing-resistant authentication is also offered, but only to the most important users. It's not obvious if the authentication requires zero knowledge. Since a private key is required by both the single authentication method and the FIDO2 essential in order to generate a nonce, the authenticator employs the corresponding public key in order to verify the signature. Most of the time, the secret key can be deduced from the mathematical link between the public key and the private key; nevertheless, doing so is frequently computationally challenging. Everything is heavily reliant on the settings and manner

[8] Users can use ordinary devices with FIDO2 to quickly authenticate to web services in desktop and mobile contexts.

used. Microsoft Azure employs Azure RBAC as its resource management system. The three basic categories of roles are owner, contributor, and reader. Additional responsibilities can be defined. This underlying concept may lead to issues if it is used in a situation where several users are storing data and wish to share, edit, or otherwise change it. The need that each group member acquires an invitation from the owner is the most onerous. With their updated I-TGDH structure, Tresorit aims to completely solve this issue by removing the need for new users to rely on the presence of any other user outside the inviter. Microsoft Azure has strong logging techniques in addition to other effective defences against potential exploitation scenarios. Users' saved data is only temporarily accessible by staff members and is immediately removed once the operation is over. Similarly, nothing may be used or copied by third-party providers. The system is vulnerable to abuse from outside law enforcement organisations despite extensive logging and stringent access rules, and it is unclear Whether or not a user is notified when a certain employee requests access to data.

3.3.3 Microsoft Azure's Encryption of Data in Motion

As with data security at rest, there are various ways to protect data when it is in use. One of the methods is TLS 1.3 and higher, as was already mentioned. By utilizing RSA-2048 encryption keys to encrypt unique keys, it becomes possible to address the issue of complete forward confidentiality. The downgrade attacks that were previously described might still be effective, depending on how they are implemented. Overall, this method of communication security is comparable to Google Cloud's modified TLS and falls short of Tresorit's Hardened TLS 1.3. The same result occurs when a TLS session member impersonates Google Cloud if the encryption is performed on the server. Simply updating all communication sessions to TLS 1.3 will fix this problem. Some users may not want this to be enforced since it forces them to invest time and money in updating their TLS, which is undesirable. TLS 1.3 will most certainly be deprecated at some point in the future due to its various issues and out-of-date algorithms, therefore this change will eventually need to be made. Another option offered by Microsoft Azure for protecting data in transit is the MAC protocol. There is also VPN security accessible. TLS and MAC are two examples of technologies that can be paired to increase security.

3.4 Amazon S3

The cloud storage component of Amazon Web Services (AWS), known as Amazon S3, is run by Amazon. Like the other two providers, AWS likewise provides a wide range of services.

3.4.1 Infrastructure Security and Other Aspects

In this instance, physical security is also present. Given that there is always a chance of specific hardware theft, this layer is just as important as the others. One of the most potent DoS attacks a single person may launch even if the attacker is unable to break the encryption and decode the data. Additionally, the offender gets immediate access to all of the equipment. As a result, Amazon continuously checks the cloud facilities in addition to keeping an eye on all activities on the property, much like the other services. Regarding storage decommissioning, Amazon complies with the National Institute of Standards and Technology (NIST) [33]. This standard is followed by Microsoft Azure as well [32], however, Google Cloud does not formally follow it although the employed policy has some similarities to the standard. Depending on the degree of security of the information stored, NIST proposes alternative courses of action (low, moderate, or high). Even though the hardware is never physically destroyed when this grade is low, data recovery is thought to be impossible. In circumstances with moderate and high ratings, hardware is always destroyed if it cannot be used again. Neither cloud provider gives specifics about how the rating is done. Data is backed up in many locations and zones so that even in the event of numerous total failures, it can be restored. AWS incorporates various components, referred to as API endpoints, to separate the internal network from the external network, offering similar benefits as the aforementioned services. Firewalls and ACL lists are strategically implemented across the network architecture to promptly detect and prohibit unauthorized entities from accessing the system. Through SSL, a remote user connects to the endpoint. To identify unusual behaviour as quickly as feasible and as effectively as possible, network activity is continuously monitored. To complete two-factor authentication, users must also input a one-time 6-digit number. Every 90 days, users must update their passwords. Additional credentials include things like access keys, key pairs, and X.509 certificates. There is no information on where or how the username and password is stored, if zero-knowledge authentication is used. However, depending on the hash technique and the hardware of the attacker, 90 days might not be sufficient to reverse the result if the password is kept conventionally. Furthermore, it is hard to guess the password within the allotted 90 days if the user chooses a very long password (100 characters). Finding an outdated password provides no helpful

information if there is no connection between it and the other data used for encryption/decryption. A hypervisor is used to virtualize the hardware in a similar way to how Microsoft Azure does [33], preventing users from accessing one another's memory, storage, and other resources.

3.4.2 Encryption at Rest in Amazon S3

Amazon S3 allows for both client- and server-side encryption. Server-side encryption in Microsoft Azure may be performed using either passwords maintained by the service providing the key management systems, keys owned by the user, or both sets of keys managed by the user and the key management solutions [33]. The first involves utilizing an AES-256 key to encrypt a single item, followed by a second layer of encryption performed by the server using a master key. The basic unit of storage in Amazon S3 is an object, which also contains some decrypted data. A bucket, a different component, is used to keep various items. This particular model provides a notably faster response compared to other services when a user requests a specific file or object. This is primarily due to the absence of a data reconstruction stage or the remarkably rapid execution of this stage. However, if an attacker lacks awareness of the keys and employs this method, they may gain access to the data more quickly and easily compared to having to restore the master key, decode the object key, and then decrypt the object itself. For external adversaries, the data reconstruction stage can serve as a significant barrier. Another option is to create a unique key for each bucket, which is utilized to generate additional keys for object encryption and undergoes cyclic rotation at predefined intervals. This approach speeds up the process significantly since all object keys are closely associated with a single key, albeit with slightly reduced security as a user master key is employed Obtaining the object-specific keys and decrypting the bucket key. In the third scenario, the services encrypt the item as previously mentioned, but the key is immediately erased from memory once encryption is complete. For decryption, the user must provide the same key. This verification step is performed by the ACL using a salted hash that was generated and recorded during the encryption process. This makes it impossible for the key management entity to recover and decrypt the key [33]. However, since the data is returned in plain text during this phase, enabling this same provider to see it, the provider may be able to study the data whenever the user request it. The fact that this idea does not provide the provider with the ability to decrypt that data whenever they want gives it various advantages over the Microsoft Azure and Google Cloud versions.

The customer has two choices when using client-side encryption: utilise a key stored in the cloud's key management service, or use and manage a locally stored key. Assuming the encryption/decryption method is AES-256, the provider may theoretically decrypt the data whenever he wants because both the key and the data are now on the cloud. There aren't many other encryption techniques, though, that provide AES's level of security, effectiveness, simplicity, and speed. Overall, the disadvantages of this case scenario are the same as those of the scenario where the provider handles the encryption and the user controls the key. As soon as the key management service on the server has authority over the keys, they will be cycled regularly. [33] states unequivocally that only the owner has access to the data that has been saved by the owner in the cloud. Additionally, the owner has the ability to establish an policy for allowing other users access to restricted resources. Access rules may be applied to users, buckets, or individual objects. While improved data management is usually good for the customer, it also introduces the risk of granting incorrect users undue privileges due to potential mistakes in policy configuration. This service also features activity monitoring at various levels, which makes it simpler to detect whether an unauthorised user accessed the data accidentally. Server-side encryption and user master keys are managed by the key management service and are used to encrypt data once more during backup. The servers also store many copies of a given item, allowing a user to restore an earlier version even if the most recent version is lost from both the backup and storage servers. If data is no longer required, the user may also archive or erase it.

3.4.3 Encryption in Transit in Amazon S3

By using Ephemeral Diffie-Hellman or Elliptic Curve Diffie-Hellman, TLS 1.3 provides flawless forward security. Additionally, each request needs to be encrypted with a distinct access key or use the AWS security token service. Although it is not advisable, TLS 1.0 is also supported [33]. A VPN connection can be made from a user, such as Google Cloud or Microsoft Azure, to the interface endpoint. Data in transit is guaranteed to be unaltered thanks to AWS Signature V4[9]. To generate the signature, customer-specific access

[9] In all AWS Regions, Amazon S3 supports Signature Version 4, a standard for verifying incoming API requests to AWS services.

keys consisting of an access key ID and a secret access key are utilized. On top of TLS, this might be considered an additional layer of security.

4. Overview

The aforementioned services may be categorized into two groups: those where the user has full control over their data but the cloud retains the ability to decrypt it at any time, and those where the provider retains unfettered access to information at any time. Either circumstance is open to abuse. In the first case, even if the cloud provider has access to the data, the cloud might be used to facilitate illegal activities. In the second scenario, the supplier has unrestricted access to the data and is free to use it in any way it pleases. It is difficult to find a medium ground where neither user is favoured because the provider's ability to analyse a file automatically means that they can duplicate the file. Since abuse is difficult to prove and the supplier has discretion over how to prevent and penalise such behaviour, this can be damaging. Several cryptographic cloud storage techniques guarantee the user complete privacy (or a high degree of it) or plans that give the provider a lot of control over the data; one example of the latter is Google Cloud's storage system. The second type is much more common because providers decide how to govern their data at their discretion. Comparing security measures for Tresorit, Google Cloud, Microsoft Azure, and Amazon S3 is shown in Table I.

Table I. Indicators of Safety: A Comparison

	Tresorit	**Google Cloud**	**Microsoft Azure**	**Amazon S3**
Encryption Method	AES256	AES256/128	AES256	AES256
Method for Sharing Data	I-TGDH	ACL	Azure RBAC	ACL
Storage Style	Tresor	Block	Block	Object/Budget
How Many Keys are Needed to Decrypt One Piece of Data?	Highly Variable	4	2	2
Security Measures for Data in Transit	Hardened TLS1.2	PFS-TLS1.2/ALTS/VPN	PES-TLS 1.2/VPN/MACsec	PFS-TLS1.2(TLS1.0 Supported)
Location of Encryption	Client-Side	Server-Side	Client-Side/Server-Side/Mixed	Client-Side/Server-Side /mixed
Key Managing Entity	Client	Server	Client /Server /Mixed	Client/Server/Mixed
Key rotation	Lazy Re-Encryption	90 days	Method Not Specified	90 days
Authentication	Zero-Knowledge	Standard/Not Mentioned	Standard/Not Mentioned	Standard/Not Mentioned
Other Security Measures for Authentication	PKI	Sign-in Location Tracking	X.509 Certificates	Signature-Based
Other Methods of Authentication	Two Factor	Two factor/FID02	Two Factor/FID02/SSO	Multi-Factor

Achieving complete prevention of a provider from utilizing the data in violation of the terms of service, while still allowing them to manage the files stored in the cloud, is practically impossible. The facilitator may also create a local copy and use it if it has the ability to read the files. The user can limit provider access by adding an additional layer of encryption to the stored files. The following are some suggested remedies for the aforementioned problems:

- **Threshold cryptography**: Threshold cryptography is one type of encryption. The allocation of the key shares will have a significant impact on the strategy's outcome. The outcome indicated above is achieved if the provider can generate the key on its own. The user may simply decline to take part if essential reconstruction is required of him. If the user chooses not to participate, the provider may threaten to take action against them. The worst punishment is to erase the user's access to the data that has been stored. However, if a user is aware of how sensitive the data being saved is, the punishment can end up helping the offender erase the evidence.
- **Storing the key on the server encrypted with a subpar encryption algorithm:** Granting the provider the capability to launch an attack if access to the files becomes necessary poses a challenge in selecting an appropriate encryption method that allows for timely execution of such actions. The supplier may receive the keys in massive quantities, creating the same issue as described earlier, if the time required to conduct the assault is too short. Finding a workable approach is made much more difficult by the gear that is available to carry out the attack.
- **Searchable encryption:** Upon initial analysis, searchable encryption appears to be a favorable choice. By employing this technique, the provider can inspect the contents of files for specific keywords,

enabling them to assess if any violations of the terms of service have occurred. Since users can either avoid using sensitive terms or use coded language, achieving this effect is much harder. Since the presence of a single trigger word is insufficient to ascertain a breach of the terms of service, a thorough review of a document may be required, which can result in a significant time investment. Files that aren't papers make this worse.This method is unsuccessful when the provider has a large amount of data saved because when searchable encryption is used on such files, reaching an exact judgement is extremely difficult and takes a very long time.

- **Asks for the user's consent:** Whenever the service provider wants to access the files, they implement this procedure. Similar to threshold cryptography, the user has the option to disengage but cannot offer a response to the request.

From the preceding solutions, we may infer the following conclusions:

- To give both the provider and the user constant file access, the document keys must be kept on the server.
- The files must only be accessible via the server.
- Users need to be notified when the server requests file access..

5. New Storage System Planned

Cocks-IBE and AES-256 CBC are two separate encryption techniques that are used in the proposed system for encrypted cloud storage to offer a high level of security and user control over their data. The public key in the Cocks IBE is the user's identity, such as their email address. Since the recipients can just use their email address as the public key to access the files, this makes it simple for users to share their files with others. IBE also enables fine-grained access controls because the encryption keys can be changed or revoked as necessary. Data is encrypted in fixed-size blocks using the widely-used AES-256 CBC encryption standard, which offers a high level of security by employing a different IV for each block. The suggested method makes sure that the IVs are secure by encrypting them using AES-256 to prevent tampering or unwanted access. A zero-knowledge authentication technique called Fiat-Shamir authentication can be used to confirm the user's identity without disclosing any personal information. This lessens the likelihood of phishing scams and other identity theft schemes. The proposed strategy also has a feature that notifies the user whenever a file inspection occurs, enabling the user to monitor who has access to their files and identify any potential misuse. The user's data may be protected from future breaches by using this function, which can help identify illegal access or suspicious activity. The proposed storage systems' pseudocode is as follows:

Pseudocode of Storage System

Step 1: Generate a private key for the user using Cocks IBE
Step 2: Encrypt the user's files using AES-256 CBC with a unique IV for each block
Step 3: Store the encrypted files and the corresponding IVs on the cloud server
Step 4: For file access:
> *Step 4.1: User sends a request for a file with their identity (email address)*
> *Step 4.2: Server retrieves the corresponding private key using the user's identity*
> *Step 4.3: Server retrieves the corresponding IV for the file*
> *Step 4.4: Server decrypts the file using the private key and IV and sends it to the user*
> *Step 4.5: Send an alert to the user that their file was accessed*

Step 5: For file upload:
> *Step 5.1: User encrypts the file using AES-256 CBC with a new IV*
> *Step 5.2: User sends the encrypted file and the corresponding IV to the server*

Step 6: For access controls:
> *Step 6.1: Users can update or revoke their private key*

Step 6.2: Server can update or revoke access to specific files

Step 7For authentication:

Step 7.1: User provides proof of identity using Fiat-Shamir authentication

Step 7.2: Server verifies the proof without revealing the user's identity

The proposed system for encrypted cloud storage that uses Cocks IBE and AES-256 CBC is generally described in the pseudocode. This pseudocode encrypts data using AES-256 CBC with a different IV for each block once the user first creates a private key for himself using Cocks IBE. The cloud server stores the encrypted files and the related IVs; when a user requests a file, the server obtains the appropriate private key and IV, decrypts the requested file, and transmits it along with a notification that the user's file was viewed. The server can update or revoke access to particular files, and the user can update or revoke their private key for access controls. Additionally, the proposed system incorporates the zero-knowledge authentication technique Fiat-Shamir, which guards against disclosing the user's identity while validating the user's identification proof.

A detailed explanation can be seen in Fig.2, the design has five active modules. Similar to an API, these modules will connect with other running modules through requests. In the proposed system, Cocks Identity-Based Encryption (IBE) would be employed to ensure that the provider notifies the user when a file is accessed and to facilitate file-sharing capabilities. To this end, cryptographic techniques will be employed. Another option is a log-based version, however, depending on the method, this variant can be subject to issues like entities fabricating logs or other similar worries. The zero-knowledge login process will leverage the Fiat-Shamir identification technique, effectively preventing the provider from attempting to decipher the password through reverse hash methods, thus safeguarding complete access to the data.Since the rewards obtained are probably not worth the effort, this is not a concern for the average user. For high-profile users who have sensitive information, this can be crucial. While the encryption and decryption will take place locally, the keys will be kept in the cloud. This approach discourages users from violating the terms of service as their activities will be regularly monitored. Furthermore, users will have visibility into whether their files have been viewed. Conversely, the provider retains continuous access to the files but is unable to do so covertly. Ideally, the terms of service should specify a regular inspection interval, such as once a week, or a predetermined number of inspections over a specified period, enabling users to promptly challenge any breaches of regulations by the provider and take appropriate action. All of this suggests that the provider may continue to use the user information to create customised profiles for manipulation, performance analysis, and ad personalisation. This will always be the situation, as was previously indicated. However, because they are unable to learn on the go, the technique used will frustrate and occasionally even slow down these entities.A local copy of the file must be made if the provider needs to have persistent access to it, which creates additional problems and costs.

While cloud components are generally perceived as reliable, an interesting aspect to note is the potential drawback of Identity-Based Encryption (IBE) where the key generation module also possesses access to the keys. Consequently, the group responsible for this module would have the ability to decrypt all stored files, which could lead to actions that violate the terms of the agreement. The design will be similar to the cloud storage systems previously discussed in that it will have a network interface to aid with traffic management. The key management entity's communications must also travel through the network interface because it is trusted yet suspicious, therefore a second encrypted connection must be established between the user and the Key Management Module (KMM). This is done to stop a module inside the local network from connecting directly to something outside of it. An AES-256 key, an IV, or the Cocks personal settings, depending on the encryption method chosen, make up the contents of messages sent via this connection. The interface shouldn't decrypt such communications. Secure key exchange protocols resistant to man-in-the-middle attacks can be employed to configure the session keys. Additionally, a separate, customized VM dedicated to traffic scanning is required for effective configuration management. While plaintext communications may be relatively brief, attackers can still attempt to directly implant malware into the key controlling unit since encrypted data cannot be adequately scanned by the network interface.

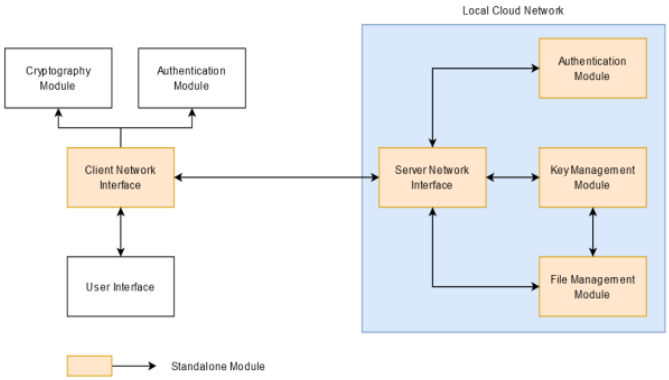

Fig.2. Proposed architecture for a storage system

Operations on the Proposed System

A user may carry out ten different actions. But before doing anything else, the user must always open an encrypted connection to the server through TLS 1.3. A more in-depth explanation of them will follow:

1. **Register:**

- To implement the authentication module, the client must first create the huge primes that exist p and q (1024 bits) from which n will be calculated. Variables s and v, each of which consists of 10 elements, are generated using the user's password as a seed, which are crucial for the login process. To ensure consistency, the login stage cannot be bypassed unless the client utilizes the password as a seed, as it would generate different values each time. Additionally, this process establishes a connection between the password, which can be alphanumeric and up to 32 characters long, and the other parameters. The creation process, which involves dealing with large primes *p* and *q*, may take up to three seconds. It is important to note that performing this process on the server could potentially be exploited as a Denial of Service (DoS) attack.
- The client directs the registration appeal, along with the numbers *n* and *v*, to the Server Network Interface (SNI).
- The SNI takes the appeal and sends it on to the CAM for further processing.
- The CAM verifies whether the user is already registered. If the user can't be located, they'll end up anywhere between n and v. The procedure's outcomes are reported to the SNI..
- If everything went according to plan, the SNI contacts the KMM.
- The user receives the user-specified separate secured connection over which the KMM outputs the Cocks private parameters.
- The user encrypts the received parameters with AES-256 and keeps them locally to ensure integrity. The alphanumeric password entered by the user will act as the encryption key.

2. **Login:**

- Both the client and the server will utilise a variable to maintain track of the user's login status. If the user is not logged in, certain client-side actions will be unavailable. This variable is also monitored by the server to make sure the client isn't engaging in improper behaviour, such as browsing other people's files.
- The client encloses his username in a login request that is sent to the SNI.
- The SNI contacts the CAM and queries the database to determine if the username is there.
- The Fiat-Shamir identification procedure is performed after the vector is sent via the SNI. carried out four more times if the answer is affirmative.
- If the result is accurate all four times, the client completes the work. If not, the login is unsuccessful.
- After a successful login, the client attempts to retrieve the previously stored secret Cocks parameters from its local storage. The user will be alerted that an integrity compromise has

taken place if the EAX decryption fails. Then, to correctly store and download files from the cloud, he must reset the settings.

3. **Logout:** To avoid synchronisation problems, the SNI maintains one thread for each additional user, while the CAM, KMM, and File Management Module (FMM) each maintain just one threat. The value of the logged-in user has changed whenever a user logs out. The thread is ended when the user logs off.
4. **View Stored/Shared Files:** The File Management Module (FMM) is responsible for monitoring owners and their respective files, while the Key Management Module (KMM) tracks files and the users authorized to access them.
5. **Reload the Cocks parameters:** This alternative will be employed in case the locally stored parameters become corrupted or compromised. Although already logged in, the client must first successfully authorise themselves.
6. **Upload a Documents:**

    - Only the file's owner has access to this setting.
    - The client notifies the SNI that he wants to save a file by sending a signal. He then produces a key and IV before using AES-256 CBC to encrypt the file. Before uploading the file for storage, the client will locally save two hashes: one of plain text and one of cypher text. The document will be encrypted if it has no metadata.
    - The client sends the file in 4096-byte chunks with a hash to guarantee its integrity.
    - Before sending the file to the FMM, the SNI waits for it to be sent successfully.
    - The SNI will then deliver the key and IV to the KMM through a different encrypted connection after receiving them from the client.
    - The KMM uses Cocks IBE to encrypt and store the key. For each additional user added to the Access List, extra keys will be produced. The KMM then relays the answer to the client and notifies the SNI of the operation's outcome.
7. **Documents Download:**

    - Only the file's owner has access to this function.
    - The client informs the SNI of the proposed action and the file to which it will be applied.
    - The SNI verifies the file is there in the KMM. If the data is accessible, it is obtained and sent to the user.
    - After making an application to the KMM module, the SNI gives the client an encrypted answer, together with the key that is encrypted and the initialization vector.
    - The client uses locally cached Cocks parameters to decode the key and IV, and then checks the ciphertext's integrity by comparing the hash to a known good one. In such a situation, the app will notify the user that the file has been re-encrypted with a new IV.

8. **Delete file:** A file can be deleted quite easily. The request is sent to both KMM and FMM after being sent to the SNI. The client is then informed of the progress of the treatment.
9. **Show or Refresh the List of Permitted File Changes:** The KMM is home to the file permissions list. Usernames are sent with activities (such as removing or adding a user through the access list) from the client. The KMM will then generate a new key for the user or remove an existing one.
10. **Upload Shared File/Download Shared File:**

    - The customer is expected to initiate contact with the SNI.
    - • This is the sequence in which the SNI transmits information to the KMM as well as FMM.
    - When a download happens, the SNI will get the file at this point. In the event of an upload, the user will just get the operation status (successful/failed).
    - • The KMM will encrypt the file again after it has been uploaded if the person requesting the upload is not the owner. The KMM then sends a request for the file to the KMM, which re-encrypts it employing the identical key but a new initialization vector (IV). The KMM then sends the key and the revised IV to the SNI, which sends them to the user through a second encrypted connection.

6. Conclusion and Future Works

In conclusion, end-to-end encryption provided by Tresorit only safeguards information kept on the company's servers; it does not shield users from phishing scams. Hackers frequently use phishing attempts to obtain user credentials, which they can then use to access the victim's data. Users need to exercise caution while entering their credentials or following links on dubious websites. Other programmes like Google Photos, which is also a well-liked data backup choice for devices running the android operating system, commonly use Google Drive (part of Google Cloud) as a digital storage service. Robust security measures are essential due to the widespread usage and high attractiveness of the service, as it becomes a lucrative target for attackers seeking to acquire specific data, including personal information. Additionally, Google Cloud provides several additional services. They implement security measures that guard against DoS assaults, and employee abuse, and lessen the severity of future data breaches by isolating the data and using multiple degrees of encryption and permissions while yet keeping the option to access the data. Overall, rather than depending on more complex cryptographic security solutions, Microsoft Azure prioritises cloud security through the use of firewalls, antimalware software, extensive network stacking, and specific cryptographic protections. This makes it possible to analyse data more quickly and with fewer resources while yet protecting yourself from outside threats. The fact that the cloud can offer such a variety of services probably influenced the decision to choose this technology. It all depends on the attacker's ability to breach the key management entity in the event of leaking. The default backup choice for Windows devices, Microsoft OneDrive, is likewise hosted by Microsoft Azure. Cloud security is even more essential given the vast number of devices running Windows. In prioritizing ease of use and availability zones, Amazon S3 emphasizes convenience and accessibility, although it is worth noting that object encryption may offer faster processing compared to block encryption, which involves the additional step of dividing the data into smaller pieces.

In the proposed system, if IBE were to be replaced with a more decentralised system in the future, user privacy would be increased because the KMM component would no longer have access to all keys. To prevent the FMM module from decoding any of the files, even if it knows all of the keys, is another strategy that might be used. Anonymizing user identities and file names so that, despite the KMM knowing which key belongs to which file, he is unable to find that particular file because it has a completely different ID inside the FMM module. This is done by keeping the lists of users and file names that are maintained within the KMM and FMM completely separate from one another. The KMM can no longer execute re-encryptions, which implies that the encryption must be carried out in another module or on the client. This is a significant disadvantage for all of these choices. This module would encounter the same problems as the KMM and provide no solutions. The user will have the option to permanently deny file analysis if encryption is performed on the client side. If the file is distributed to many people, another issue might develop. Only the owner will be able to tell if his file has been accessed. He has no notion whose user acted. At first, glance, associating an IV with a particular identity seems to solve the issue; nonetheless, it is advised against repeatedly employing the same IV when using AES-256 CBC. The user and KMM module may share a pseudo-random number generator that generates IVs based on a string. The username and the current date will be included in the string. This solution demands that the cloud maintain accurate logs. The full indent version of the Boneh-Franklin IBE scheme can be replaced with Cocks IBE if IBE is kept and a solution to the aforementioned issues cannot be discovered. Block storage would become far more effective as a result. Although Cocks IBE can be used for block storage as well, key storage consumes a lot of space since block storage needs many keys, each of which has an after-encryption size of 136KB for AES keys.

Funding: This research received no external funding.
Conflict of interest: The authors declare no conflicts of interest.

References

[1] E. Goh, H. Shacham, N. Modadugu, and D. Boneh, “SiRiUS: Securing remote untrusted storage,” in *Netw. Distrib. Syst. Secur. (NDSS ’03) Symp.*, 2003, no. 0121481, pp. 131–145. Accessed: Jan. 19, 2023. [Online]. Available: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=9779af680a9171f44c9085cc76b6878a906d4260

[2] R. A. Popa, J. R. Lorch, D. Molnar, H. J. Wang, and L. Zhuang, “Enabling security in cloud storage SLAs with CloudProof,” in *Proceedings of the 2011 USENIX Annual Technical Conference, USENIX ATC 2011*, 2019, pp. 355–368. Accessed: Jan. 19, 2023. [Online]. Available: https://www.usenix.org/events/atc11/tech/final_files/Popa.pdf

[3] A. Shraer, C. Cachin, A. Cidon, I. Keidar, Y. Michalevsky, and D. Shaket, “Venus: Verification for untrusted cloud storage,” in *Proceedings of the ACM Conference on Computer and Communications Security*, 2010, pp. 19–29. doi: 10.1145/1866835.1866841.

[4] K. D. Bowers, A. Juels, and A. Oprea, “HAIL: A high-availability and integrity layer for cloud storage,” in *Proceedings of the ACM Conference on Computer and Communications Security*, 2009, pp. 187–198. doi: 10.1145/1653662.1653686.

[5] M. T. Goodrich, C. Papamanthou, R. Tamassia, and N. Triandopoulos, “Athos: Efficient authentication of outsourced file systems,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2008, vol. 5222 LNCS, pp. 80–96. doi: 10.1007/978-3-540-85886-7_6.

[6] D. K. Gifford, P. Jouvelot, M. A. Sheldon, and J. W. O'Toole, "Semantic file systems," *ACM SIGOPS Oper. Syst. Rev.*, vol. 25, no. 5, pp. 16–25, Sep. 1991, doi: 10.1145/121133.121138.

[7] B. Gopal and E. Al, "Integrating content-based access mechanisms with hierarchical file systems," *OSDI '99 Proc. 3rd USENIX Symp. Oper. Syst. Des. Implement.*, 1999, Accessed: Jan. 19, 2023. [Online]. Available: https://www.usenix.org/conference/osdi-99/integrating-content-based-access-mechanisms-hierarchical-file-systems%5Cnpapers2://publication/uuid/C1601F2A-AE83-493B-BBAE-69D1552F78F0

[8] A. Leung, A. Parker-Wood, and E. L. Miller, "Copernicus: A Scalable, High-Performance Semantic File System," 2009. Accessed: Jan. 19, 2023. [Online]. Available: http://www.ssrc.ucsc.edu/

[9] M. Seltzer and N. Murphy, "Hierarchical file systems are dead," in *Proceedings of HotOS 2009 - 12th Workshop on Hot Topics in Operating Systems*, 2009.

[10] M. Kallahalla, E. Riedel, R. Swaminathan, Q. Wang, and K. Fu, "Plutus: Scalable secure file sharing on untrusted storage," in *Proceedings of FAST 2003: 2nd USENIX Conference on File and Storage Technologies*, 2003, pp. 29–42. Accessed: Jan. 19, 2023. [Online]. Available: https://www.usenix.org/event/fast03/tech/full_papers/kallahalla/kallahalla_html/

[11] K. Fu, S. Kamara, and T. Kohno, "Key regression: Enabling efficient key distribution for secure distributed storage," *Comput. Sci. Dep. Fac. Publ. Ser.*, no. February, p. 149, 2006.

[12] R. A. Popa, J. R. Lorch, D. Molnar, H. J. Wang, and L. Zhuang, "Enabling security in cloud storage SLAs with CloudProof," in *Proceedings of the 2011 USENIX Annual Technical Conference, USENIX ATC 2011*, 2019, pp. 355–368.

[13] D. Mazières and D. Shasha, "Building secure file systems out of byzantine storage," in *Proceedings of the Annual ACM Symposium on Principles of Distributed Computing*, 2002, pp. 108–117. doi: 10.1145/571825.571840.

[14] O. Goldreich and R. Ostrovsky, "Software protection and simulation on oblivious RAMs," *J. ACM*, vol. 43, no. 3, pp. 431–473, May 1996, doi: 10.1145/233551.233553.

[15] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy*, 2000, pp. 44–55. doi: 10.1109/secpri.2000.848445.

[16] E. J. Goh, "Secure Indexes," *IACR Cryptol. ePrint Arch.*, pp. 1–19, 2003.

[17] M. Chase and S. Kamara, "Structured encryption and controlled disclosure," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2010, vol. 6477 LNCS, pp. 577–594. doi: 10.1007/978-3-642-17373-8_33.

[18] J. Baek, R. Safavi-Naini, and W. Susilo, "Public key encryption with keyword search revisited," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2008, vol. 5072 LNCS, no. PART 1, pp. 1249–1259. doi: 10.1007/978-3-540-69839-5_96.

[19] G. Ateniese *et al.*, "Provable data possession at untrusted stores," in *Proceedings of the ACM Conference on Computer and Communications Security*, 2007, pp. 598–610. doi: 10.1145/1315245.1315318.

[20] A. Juels and B. S. Kaliski, "Pors: Proofs of retrievability for large files," in *Proceedings of the ACM Conference on Computer and Communications Security*, 2007, pp. 584–597. doi: 10.1145/1315245.1315317.

[21] H. Shacham and B. Waters, "Compact proofs of retrievability," *J. Cryptol.*, vol. 26, no. 3, pp. 442–483, 2013, doi: 10.1007/s00145-012-9129-2.

[22] G. Ateniese, R. Di Pietro, L. V. Mancini, and G. Tsudik, "Scalable and efficient provable data possession," in *Proceedings of the 4th International Conference on Security and Privacy in Communication Networks, SecureComm'08*, 2008. doi: 10.1145/1460877.1460889.

[23] Y. Dodis, S. Vadhan, and D. Wichs, "Proofs of Retrievability via Hardness Amplification," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2009, vol. 5444 LNCS, pp. 109–127. doi: 10.1007/978-3-642-00457-5_8.

[24] G. Ateniese, S. Kamara, and J. Katz, "Proofs of storage from homomorphic identification protocols," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2009, vol. 5912 LNCS, pp. 319–333. doi: 10.1007/978-3-642-10366-7_19.

[25] C. C. Erway, A. Küpç;ü, C. Papamanthou, and R. Tamassia, "Dynamic provable data possession," in *ACM Transactions on Information and System Security*, Apr. 2015, vol. 17, no. 4. doi: 10.1145/2699909.

[26] Y. Kim, A. Perrig, and G. Tsudik, "Tree-based group key agreement," *ACM Trans. Inf. Syst. Secur.*, vol. 7, no. 1, pp. 60–96, Feb. 2004, doi: 10.1145/984334.984337.

[27] I. Lám, S. Szebeni, and L. Buttyán, "Invitation-oriented TGDH: Key management for dynamic groups in an asynchronous communication model," in *Proceedings of the International Conference on Parallel Processing Workshops*, 2012, pp. 269–276. doi: 10.1109/ICPPW.2012.40.

[28] F. Cuppens, N. Cuppens-Boulahia, and J. Thomas, "S-TGDH, secure enhanced group management protocol in ad hoc networks," *Cris. Int. Conf. Risks Secur. Internet Syst.*, p. 8p., Jul. 2007, Accessed: Jan. 19, 2023. [Online]. Available: https://hal.science/hal-00426444

[29] "Method and system for handling of group sharing in a distributed data storage, particularly in P2P environment," Mar. 2012.

[30] "Cloud Storage Security - Secure Cloud Storage from Tresorit." https://tresorit.com/security#encryption (accessed Jan. 19, 2023).

[31] G. Cloud, "Google Cloud Security Whitepapers Encryption at Rest in Google Cloud Encryption in Transit in Google Cloud Application Layer Transport Security in Google Cloud," 2018.

[32] Microsoft, "Azure security fundamentals documentation," *Microsoft Website*, 2021. https://learn.microsoft.com/en-us/azure/security/fundamentals/ (accessed Jan. 19, 2023).

[33] A. Web Services, "A r c h i v e d Amazon Web Services: Overview of Security Processes," 2020, Accessed: Jan. 19, 2023. [Online]. Available: https://aws.amazon.com/