Attacks on MEGA Contact Relationships

by

Ken J. Kiisa

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Monday January 12, 2026 at 10:00 AM.

Student number:	5446333	
Project duration:	April 30, 2025 – January 12, 2026	
Thesis committee:	Prof. dr. G. Smaragdakis,	TU Delft,
	Dr. K. Liang,	TU Delft, Supervisor,
	Dr. J.E.A.P. Decouchant,	TU Delft
Daily Co-Supervisor:	Dr. H. Chen	TU Delft

TU Delft

Attacks on MEGA Contact Relationships

Ken J. Kiisa
Technical University of Delft
K.J.Kiisa@student.tudelft.nl

Dr. Kaitai Liang
Technical University of Delft
Kaitai.Liang@tudelft.nl

Dr. Huanhuan Chen
Technical University of Delft
H.Chen-2@tudelft.nl

ABSTRACT

MEGA is a popular cloud storage provider in both commercial and consumer markets [2][1]. MEGA claims to provide secure storage, in a threat model where even the storage provider should be unable to tamper with a user's data undetected [5]. Previous work by Backendal et. al., as well as other follow-up research works, discovered several attacks that an adversarial storage provider could perform to covertly read and write a user's storage [7]. MEGA's patches to the attacks solve the initial attacks that allow for the attack chain to take place, but did not solve the fundamental problems in the security architecture that enabled these attacks [6]. This work provides 5 attacks on user's contact relationships and folder sharing, that even after the patches, allow for an adversarial storage provider to manipulate a user's contact list, and forge data in their secure storage.

1 INTRODUCTION

The Cloud Storage Services market has become very large in the past decade. In 2023, the size of the global consumer cloud storage market was measured to be 16.43 billion USD, and is projected to grow to 70.2 billion USD by 2032 [9].

One of these cloud providers is MEGA, a secure cloud storage provider with the claim of providing secure cloud storage with zero-knowledge encryption, providing a security model meant to be secure against even an adversarial storage provider [2]. They are a very large player in the space, boasting over 10 million active daily users, with over 335 million unique registered users worldwide [1]. Their security model is intended to provide security such that MEGA is only trusted to provide availability, and would otherwise be unable to read or manipulate any of the data they are storing on your behalf. This is a very strong security model, but consistent with their claims on the level of security they offer:

"Because of this, no third party—including MEGA—can view the data's contents. This approach ensures that all communication remains private and secure, protecting both confidentiality and integrity"[5]

In 2023, research by Backendal et al. found several attacks on MEGA's secure storage architecture, enabling an adversarial server provider to discover a user's private RSA key, decrypt their files, forge new files and to eavesdrop on communication with other users [7]. They were able to do this via an attack, where modifications to the user's encrypted RSA key would leak information about it, enabling an adversary to learn the full RSA key after 512 log in attempts.

The attack for forging data only required that the attacker was aware of an arbitrary plaintext/ciphertext pair, encrypted with the user's master key. With this known pair, a file encryption can be forged with the plaintext as a key, and placed into user storage, without a user being able to prove that they were not the one to place that file in their storage. As such, this means that an attacker's ability to create forged data for a user's private storage depends on their ability to discover arbitrary plaintext/ciphertexts pairs. Previous attacks that could be used to discover those plaintext/ciphertext pairs have been patched, but the integrity vulnerability itself remains [7][6].

Subsequent patches by MEGA prevented the RSA key recovery attacks which enabled the integrity attack (among others), however the intervention did not fix the other attacks, as those would require much deeper reworking of MEGA's security architecture. Such a rework would require replacing established encryption algorithms and introducing new keys for key separation, such that a majority of existing user data would need to be re-encrypted to enforce key separation [6] [13] [10].

In this paper, we present 5 novel attacks on MEGA's services, that allow an attacker to perform the storage integrity attack proposed in [7] in spite of the patches that MEGA has put in place, as well as to read and manipulate any and all communication between two users on MEGA. The attack re-enable the integrity attack from work by Backendal et al. by providing methods to discover new plaintext/ciphertext pairs that could be used in the integrity attack. The attacks on communication use the lack of integrity protection on the main signature key a user uses to authenticate themselves to contacts, to allow an attacker to replace it without a user being able to detect the change.

2 CONTRIBUTIONS

After each publishing of previous attacks, MEGA made patches to prevent the proposed attacks from being performed [7][6]. This often involved quick and dirty interventions that blocked the first attack in the proposed kill-chain, but did not fix the fundamental security issues in the architecture. Namely, many of the attacks were possible due to a lack of integrity protected encryption and key-separation between unrelated parts of MEGA's security architecture, that allowed small vulnerabilities in one system to enable a much more invasive attack in another. Those two fundamental issues were not tackled, because properly securing the system in a way that would also prevent future attacks from being discovered with the same techniques would require the usage of AES-ECB to be replaced with AES-GCM, and for new keys to be introduced in order to enforce strict key separation between disparate parts of the architecture. This would have required the vast majority of

Msc. Computer Science Thesis, January 2026, TU Delft, Delft, Netherlands

existing user data to be re-encrypted, which might be the reason the measure was not taken. However, patches were applied that did prevent the initital attacks in each paper as written, meaning that it would not be possible to perform the series of attacks in each paper without modifications or discovery of novel attacks.

For example, when the initial RSA key recovery attack was discovered by Backendal et al., MEGA implemented a system for sanitization and integrity checks for RSA keys to prevent tampering. This patch prevented the initial attack in the proposed kill-chain, which meant that the rest of the attacks proposed in the paper would not be possible to perform, in spite of being left unpatched, because the first attack that is required to be performed to enable them has been secured [7]. However, the other attacks did not have further protections implemented to prevent them in isolation. Follow up research was sometimes able to re-enable previous attacks in spite of the patches, for example in [6], where an RSA key recovery attack was rediscovered after the first one was patched in [7].

Currently, attacks that have been proposed in previous works cannot be performed as written, as they depend on an attack in their paper that has been patched. This paper presents 5 novel attacks on MEGA's services. These are attacks that can still be performed after the patches that MEGA has applied to prevent attacks Attacks in previous works. These attacks also re-enable the storage integrity attack proposed in [7], even after attacks in previous works that enabled it were patched out. These novel attack are namely:

(1) Encrypted Contact Fingerprint Plaintext/Ciphertext Pair Recovery
(2) Ed25519 Signature Key Replacement
(3) First-Contact Establishment Covert Ed25519 Key Replacement
(4) Shared Folder Node Key Private Encryption Matching
(5) Shared Folder Invitation Encryption Oracle

Previous works manipulated storage and the Bleichenbacher attack in [7] allowed an adversary to perform a man in the middle (MITM) attack on MEGAChat communication, but attacks that allow manipulation of any communication between MEGA accounts have not been proposed. Our proposed series of attacks on user contact relationships allow an adversarial server provider to read and manipulate all communication between two users, as well as manipulating the user contact lists.

3 ETHICAL CONSIDERATIONS

Before the publication of this paper, MEGA was emailed on 19-09-2025 about the potential attacks described here. After receiving no response, they were later contacted again on 19-12-2025. A response was received, but no confirmation has been given yet on the presence of these attacks, nor a notice to maintain confidentiality for a set period of time. As of today, they are processing the bug ticket.

The attacks described in this paper were implemented on a Proof of Concept simulation of a MEGA application interaction. These attacks do not contain actual code from MEGA's server side, as that is not publically available information. The User side code is based off of publically available specifications, namely the MEGA 2022 June Security Whitepaper [12], and the MEGA Client Application SDK [3]. The attacks as implemented in the Proof of Concept, do not involve MEGA's servers, actual users or user accounts, or any of MEGA's commercial client software. The simulation is developed as Python objects interacting with each other, without involving MEGA's commercial client software.

4 PREVIOUS WORK

There is existing work on the topic of MEGA's security. This work builds on the research conducted by Backendal et. al. in [7]. In that work, a detailled explanation of MEGA's security architecture is given, along with its main contribution being several attacks, including an RSA key recovery attack, a plaintext-recovery attack, a secure storage integrity attack, and a Bleichenbacher communication attack.

Two relevant user encryption keys in the work by Backendal et al. are the user RSA communication key and the user master key, k_M. The RSA key is used to encrypt blocks of data that are sent to users, either by MEGA or by other users in their contact list. The master key is an AES-128 key, that is used to symmetrically encrypt all keys that are used for different purposes in MEGA's services, including the RSA communication key. In MEGA's architecture, the mathematical arguments that make up the RSA private key, **q, p, d and u**, are individually encrypted (along with their bit lengths) using AES-128 in Electronic Code Book, which means they lack integrity protection.

The RSA key recovery attack exploited this by repeatedly modifying the value u, such that the presence or lack of an error from the user side would incrementally inform an adversary of the exact value of the private key over the course of 512 log in attempts [7]. Successfully performing this attack would then allow an adversary to manipulate messages sent to the user, such that a master key decryption oracle emerges in the plaintext recovery attack, by selectively inserting two AES blocks of ciphertext data into every log in attempt by the user, the session ID value returned would contain the corresponding plaintext data. Using that decryption oracle, the keys with which data in secure storage was encrypted, alongside other arbitrary ciphertexts, could be recovered as long as it was encrypted by the master key. The integrity attack used the ability to decrypt and re-encrypt the storage keys as the basis for modifying secure storage, while also showing that knowledge of select plaintext/ciphertext pairs could allow an adversary to place new files into a user's storage without their knowledge, and without traces of tampering. As a result of that research, MEGA implemented patches to prevent the RSA key recovery attack, by implementing client checks on the format of RSA keys it receives [7] [6]. This prevented the other attacks in the paper that depended on the RSA key recovery attack from being performed. However, each of those other attacks in isolation was left unpatched.

After the publishing of that paper, further research building on the work by Backendal et al. was conducted into MEGA's vulnerabilities. Research by Heninger et. al. in [10] reduced the number of required log in attempts for the unpatched version of the RSA key recovery from 512 to 6, by exploiting the additional 43 bytes of data that each failed log in attempt provides the adversary. The version being unpatched meant that the attack could not be implemented in practice since the active version of MEGA's services at the time already had patches installed that prevented the attack. Some of

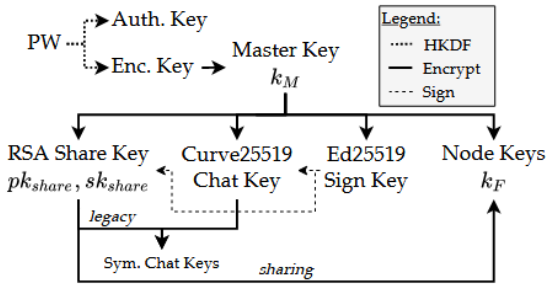

Figure 1: Diagram of keys used in MEGA's services and their relationships, sourced from [7].

the researchers from [7] then performed follow-up research in [6], and used the error handling and sanity checks implemented in the patches to [7], along with a new encryption oracle enabled by the MEGADrop feature, to discover a new RSA key recovery attack that worked after the patches to attacks in previous work. This new RSA key recovery attack also only required two log in attempts to successfully recover the key. This new attack, along with the encryption oracle, has since been patched by MEGA.

Later work by Shimoe et. al. in [13] developed an improvement for the RSA key recovery attack, building on the work in [10] to propose an attack that could recover the RSA Private key in 4 log in attempts, without needing the encryption oracle used in [6]. This attack exists for the unpatched version of MEGA in which the work by Heninger et al. was conducted, meaning that this was also an attack that could not be performed in practice.

5 MEGA'S ARCHITECTURE

This section gives an overview of the components of MEGA's security architecture that are relevant to the attacks proposed in this paper. For a more comprehensive overview of MEGA's security architecture, an explanation is given in [7] and [12].

5.1 Notation

First, we shall establish notation that will be used in the rest of the paper.

- The encryption of a 128-bit AES plaintext block pt by key k is denoted by $[pt]_k$;
- Encryption using some other encryption algorithm $Algo$, of a plaintext block pt using key k, is denoted by $Algo(pt, k)$; Similarly, Hashing using some hashing algorithm $HAlgo$ is denoted by $HAlgo(pt)$;
- 0^x denotes a 0-block of x bits long;
- Array-like referencing of an index i in an object A is written with $A[i]$, with ranges of indices starting from i and ending in $j-1$ denoted by $A[i:j]$. For example, for $a = 011001010$, $a[1:5] = 1100$;
- The concatenation of two blocks of data, e.g. a and b, is denoted by $a\|b$;
- $\otimes$ indicates a XOR-operation;
- k_M refers to the user master key.

5.2 The Architecture

A diagram of MEGA's encryption key architecture is shown in figure 1. Whenever a user logs in to MEGA, their password is used to generate an authentication key and an encryption key. The authentication key is used to authenticate the user's identity to MEGA, upon which MEGA will send the encrypted master key, k_M, to the user. The user will then decrypt and retrieve their master key, using the previously generated encryption key. At the core of MEGA's security architecture lies the AES-128 Master Key k_M that encrypts all other keys used for different services provided by MEGA.

One of those services that MEGA provides is secure communication between accounts. In order to secure that communication, each user generates one set of several asymmetric encryption keys for separate purposes, which are the:

- RSA Communication Key
- Curve25519 Chat Key
- Ed25519 Signature Key

The RSA communication key is used for encrypting and sending of data, either from MEGA or a user's contacts, to the user who owns the key. The Curve25519 chat key is used by users to exchange keys for the MEGA chat feature with eachother. The users use eachothers public Curve25519 chat key in combination with their own private Curve25519 key in order to generate the identical secure symmetric keys. The last asymmetric key, the Ed25519 Signature, is used as a trust root for the other two keys. The signature key signs the public keys of the other two asymmetric keys, so that a user can verify that the key they received from their contact is correct. This leaves the integrity of the Ed25519 key as the most important for ensuring that contact keys have not been tampered with. For the three asymmetric key pairs above, the private keys are encrypted using AES Electronic Code Book mode with the user's master key, while the public keys are stored unencrypted, and signed with the Ed25519 Signature Key. The private keys are not signed.

Electronic Code Book mode (ECB) means that the encryption scheme directly encrypts each datablock individually with the AES key, instead of using methods such as for example a nonce (number used once) to add integrity protection to encryption.

Whenever a user logs into MEGA, they retrieve all keys associated with their account, decrypt them, and then verify them against each other to detect any omission or manipulation by the server or MITM attackers.

5.3 File Storage

Secure storage is handled in a node tree structure, with leaf nodes being files, and all non-leaf nodes being folders, much like with regular computer file system storage. A diagram of an example file tree can be found in figure 2. The contents and attributes of nodes are encrypted using a custom implementation of AES-CCM (Counter with CBC-MAC), with a fresh AES Node key that is generated for each node. That Node key is then "obfuscated" with the Nonce used in the AES-CCM encryption, and a MAC generated from the ciphertext blocks of the node. The structure of the obfuscated node key is shown in figure 3. That obfuscated node key is then encrypted using AES-ECB with the user's master key, and stored next to the encrypted node.

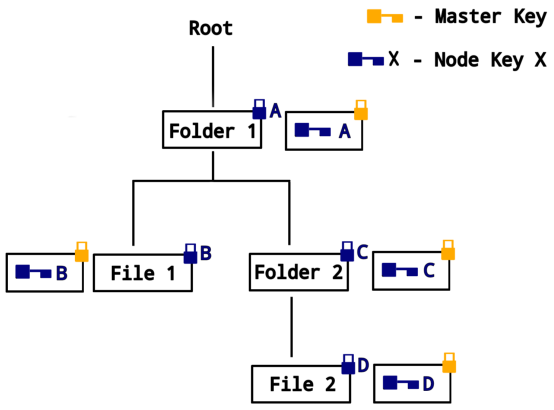

Figure 2: Example node tree structure.

Obfuscated File Key = [
File Key[0] ⊕ IV[0],
File Key[1] ⊕ IV[1],
File Key[2] ⊕ Condensed MAC[0] ⊕ Condensed MAC[1],
File Key[3] ⊕ Condensed MAC[2] ⊕ Condensed MAC[3],
IV[0],
IV[1],
Condensed MAC[0] ⊕ Condensed MAC[1],
Condensed MAC[2] ⊕ Condensed MAC[3]
];

Figure 3: Structure of 256 bit obfuscated file key. Each section in the array is 32 bits. Sourced from [12].

5.4 Contact Relationships

When two users wish to establish a contact relationship, a contact request is sent from one to the other. We will denote the users as A and B. When A receives a contact request from B, A pulls the unencrypted asymmetric public keys and public key signatures of B from MEGA. These include the following:

- Ed25519 Signature Key
- RSA Communication Key
- RSA Communication Key Signature
- Curve25519 Chat Key
- Curve25519 Chat Key Signature

Every account stores the public keys associated with their account unencrypted. The public Ed25519 Signature Key $k^{Public}_{Signature}$ is considered the trust root key of the user B. Upon verifying B's signatures with the given signature key, A creates a fingerprint of the signature key and encrypts it using AES Galois-Counter Mode (GCM) [11], as follows:

$$\text{Fingerprint} = \text{SHA-256}(k_{Signature})[0{:}160] \| 0^{96}$$
$$N, CT, Tag = \text{AES-GCM}(Fingerprint, k_M)$$

The 12 byte nonce N, ciphertext CT, and Tag are then sent to MEGA for storage, to check for future tampering of contact public keys.

Galois Counter Mode is an encryption scheme that provides integrity protection to encryptions, by generating both the ciphertext and a unique tag from it that can only be generated and verified using the encryption key [11]. By using this, it becomes much more difficult to manipulate a ciphertext without the means to generate a valid tag. This is important in this case, because as $k^{Public}_{Signature}$ is an unencrypted value, meaning that anybody who can read the key can generate the fingerprint and thus know both the ciphertext and plaintext for the encryption. The tag is used to prevent manipulation of the fingerprint in storage, however, the knowledge of the plaintext/ciphertext pair still poses problems elsewhere in the architecture.

6 ATTACKS

6.1 Encrypted Contact Fingerprint Plaintext/Ciphertext Pair Recovery

6.1.1 Threat Model. This attack assumes an adversary that is able to read the unencrypted communication between MEGA and the user, that MEGA would be able to read. The adversary is assumed to be able to freely retrieve the unencrypted public asymmetric keys associated with each account, but without knowledge of the user's master key. The goal of the adversary is to retrieve a plaintext/ciphertext pair encrypted under a user's master key.

6.1.2 Attack Description. For the counter mode part of AES-GCM encryption, the nonce N itself is not used as the counter, but instead a byte-sequence J_0, derived from the nonce N [11]. In the case that the nonce N is 12 bytes long, $J_0 = N \| 0^{31} \| 1$. In the case that N is not 12 bytes long, then according to the AES-GCM specification, $J_0 = \text{GHASH}([0^{128}]_{k_M}, N)$. GHASH is a special hashing algorithm developed for the GCM encryption mode. Because MEGA enforces that nonces for AES-GCM encryption must be 12 bytes long, that means that an adversary can derive J_0 without $[0^{128}]_{k_M}$. This allows for the following attack:

$$CT = \text{AES-GCM}(Fingerprint, k_M)['CT']$$
$$= ([J_0 + 1]_{k_M} \oplus \text{Fingerprint}[0{:}128]) \|$$
$$([J_0 + 2]_{k_M} \oplus \text{Fingerprint}[128{:}256])$$
$$= ([J_0 + 1]_{k_M} \| [J_0 + 2]_{k_M}) \oplus Fingerprint$$

Since the public Ed25519 key is stored unencrypted, the adversary also knows it. The adversary can calculate the fingerprint of the Signature key in the same way as A, and retrieve the AES-ECB plaintext/ciphertext pairs $(J_0 + 1, [J_0 + 1]_{k_M})$ and $(J_0 + 2, [J_0 + 2]_{k_M})$. This attack could then likewise be performed on B, when it receives A's public keys.

The sequence of actions in the attack can be seen in figure 4.

6.1.3 Complexity. This attack takes O(1) time to perform per contact establishment. The processing requires a single SHA-256 hashing of a 256 bit block.

6.1.4 Consequences. As much of MEGA's security architecture is built around keys encrypted with AES-ECB without integrity checks, arbitrary plaintext/ciphertext pairs allow for integrity attacks, including key replacement attacks. In [7] an integrity attack

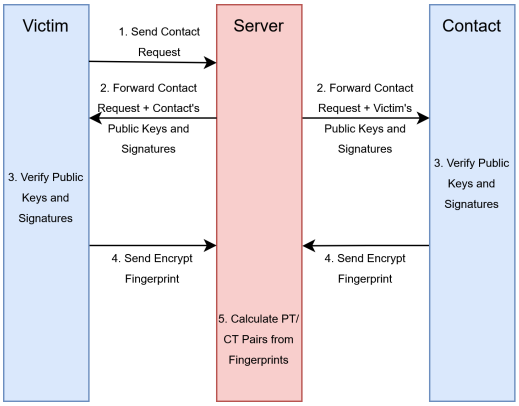

Figure 4: Sequence of actions taken by Victim, Contact and Server in Encrypted Contact Fingerprint Plaintext/Ciphertext Pair Recovery attack.

is described where a plaintext/ciphertext pair can be used to forge an encrypted file in a user's encrypted storage.

6.2 Ed25519 Signature Key Replacement

6.2.1 Threat Model. This attack assumes an adversary who has taken control of MEGA's systems. It also assumes that the adversary has at least one plaintext/ciphertext pair $(N, [N]_{k_M})$ for an arbitrary plaintext, encrypted with the user's master key. An assumption is made that the victim, when logging out, does not store a local copy of their asymmetric keys, e.g. this user exclusively uses the web version of MEGA's services. The adversary is assumed to be able to freely retrieve and edit the asymmetric keys associated with each account, but without knowledge of the master key. The goal of the adversary is to replace the Ed25519 signature key of a user, in a way that a user cannot detect.

6.2.2 Attack Description. On the first time that a new account logs in, it generates and stores the following asymmetric keys (encryptions) and signatures on MEGA's servers:

- Ed25519 Signature Public Key
- Encrypted Ed25519 Signature Private Key
- RSA Public Key
- RSA Public Key Signature
- Encrypted RSA Private Key
- Curve25519 Public Key
- Curve25519 Public Key Signature
- Encrypted Curve25519 Private Key

All of the private keys are encrypted with AES-ECB, using the user's master key, without integrity protection. Ed25519 Private Keys are 256 bits. This means that the Ed25519 Private Key Encryption can be replaced by the server, and as long as the Ed25519 Public and other key signatures match, the replacement will be accepted by the client.

Assuming a single plaintext/ciphertext pair $N, [N]_{k_M}$, the server can replace the stored Ed25519 key with $k^{*Public}_{Signature} = [N]_{k_M} \| [N]_{k_M}$, which the user would decrypt to $N \| N$. Using this, a new Ed25519 public key can be generated from the plaintext $N \| N$, and used to replace the current one. From here, because the RSA and Curve25519 public keys and signatures are stored unencrypted, the signatures can be replaced with signatures generated by the new private Ed25519 key $k^{*Public}_{Signature}$.

When the victim on a subsequent login pulls all of these keys, the signatures will verify, and they will not be able to tell that any of the keys were tampered with.

6.2.3 Complexity. This attack takes O(1) time to complete for a given user. The execution time is bounded by the time it takes to concatenate two 128 bit blocks, and to overwrite the key in storage.

6.2.4 Consequences. This attack gives an adversary the ability to forge signatures with a key that the victim would believe is secure. The signature key is used to sign public keys, which are used for communication with the victim's account. This allows an adversary to perform MITM attacks on this communication.

By having the server send the contact a different known RSA key than what the victim uses, signed with the replaced signature key, the contact would believe that the key is valid, and encrypt messages and files using it. The adversary could then decrypt and read the contents before re-encrypting it with the victim's actual RSA key and forwarding the message to them.

In the case of a folder being shared between users, this allows an adversary to decrypt the share key for that folder, which would allow them to read all files and make any changes to the contents that the adversary wants, removing integrity and secrecy from the shared secure folder.

6.2.5 Limitations. The user who was attacked will not be able to tell that their Ed25519 key has been replaced, but any of their contacts that were established before this attack will notice. When a contact pulls the victim's public keys, the Ed25519 public key will be compared to the encryption of the fingerprint of it from before the attack. The mismatch will trigger an alert in the client program, and notify the user that the key was changed. They could then contact the victim and alert them that their key was changed.

This attack on its own does not permit the replacement of the RSA and Curve25519 keys, but it does allow MEGA to at any point forge signatures for a new RSA or Curve25519 that is then given to the victim's contacts.

When the victim looks at its Ed25519 private key, then it may be alerted to tampering by seeing that the structure follows $N \| N$, which would be a highly unlikely to generate pattern. This alert can be avoided by having a second arbitrary plaintext/ciphertext pair $M/[M]_{k_M}$, s.t. $N \neq M$, and replacing either the first or second AES block with $[M]_{k_M}$.

If the victim stores a copy of their Ed25519 key locally, either manually, or via the application they use for MEGA's services doing this automatically, they will immediately be alerted of tampering of the key when comparing them with what the server fetched.

6.3 First-Contact Establishment Covert Ed25519 Key Replacement

6.3.1 Threat Model. This attack assumes an adversary that has taken over MEGA's systems. The adversary is assumed to be able to freely retrieve and edit the asymmetric keys associated with each account, but without knowledge of the master key. The goal of the adversary is to replace the Ed25519 signature key of a user, in a way that a user and their contacts cannot detect.

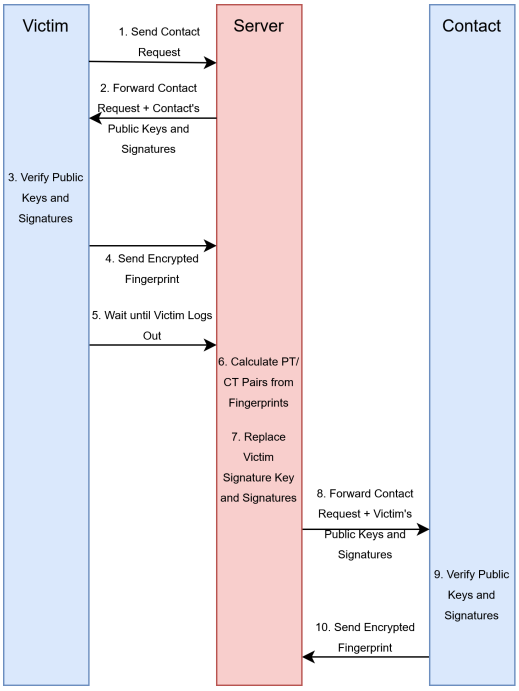

Figure 5: Sequence of actions taken by Victim, Contact and Server in First-Contact Establishment Covert Ed25519 Key Replacement attack.

6.3.2 Attack Description. In the Ed25519 replacement attack, one of the main limitations is that if a user has previously established contacts before the attack, they will be notified that the signature was changed. This sequence allows every fresh account on MEGA to be vulnerable to having their Ed25519 key replaced, without any of their contacts being able to verify if it has been.

Take two users A and B, where A is establishing with B the first contact on A's account. If the server first sends the public keys and signatures of B to A, A will accept and encrypt the fingerprint of B's Ed25519 key, before sending it back to the server. The server could then perform the first PT/CT recovery attack, retrieving a known plaintext/ciphertext pair. The server could then perform the Ed25519 Key Replacement Attack with the given ciphertext pairs.

The server will delay sending A's public keys to B, until the Ed25519 key is replaced, to avoid B detecting the change in signature key. When the Ed25519 replacement attack has been performed, then A's changed public keys will be sent to B.

The sequence of actions in the attack can be seen in figure 5.

6.3.3 Complexity. This attack takes O(1) time to complete for a given user.

6.3.4 Consequences. This attack means that any fresh account on MEGA could have their communication keys' integrity breached on the first contact establishment. A's first contact, B, would not know that A's Signature key was changed, and any future contacts would not be able to know either.

Effectively, MEGA would be capable of performing MITM attacks on all communication done between any users using MEGA's services, as long as this attack was performed on the user's first contact relationship. This is without any prior knowledge or a user performing any other actions before establishing their first contact relationship.

6.3.5 Limitations. If the victim does not log out and back in while the Ed25519 replacement attack occurs, then at any time where the Ed25519 key becomes relevant, the victim will continue to use the signature key they had before. In order to avoid B being alerted to the key change before the victim starts using the new key, any signatures that were made with the old key, would need to be resigned with the new key by the adversary.

The same limitation applies as for Ed25519 key replacement, that if the user has a local copy of the key, they will be alerted to tampering when comparing the keys.

It is not possible to perform this attack on both A and B without either user being alerted to tampering. This is because in order to replace the signature key of one of these users, the other's public keys must be sent to the victim first. If the attack is performed on A, and then the attack is performed on B, then A would be notified at a later login that B's Ed25519 key was changed.

6.4 Shared Folder Node Key Private Encryption Matching

6.4.1 Threat Model. This attack assumes an adversary that has taken control of MEGA's servers, and that has an active MITM attack ongoing between the involved users A and B. The adversary is assumed to have a copy of user A's storage before the start of the attack. The goal of the adversary, is to retrieve plaintext/ciphertext pairs encrypted under user A's master key.

6.4.2 Attack Description. In MEGA, it is possible to turn a previously private folder full of data in a user's storage into a shareable folder. This involves generating a new symmetric AES share key, and re-encrypting all of the obfuscated node keys in the folders' files and subfolders. In this process, the underlying node key is not replaced, so the data and header ciphertexts remain unchanged.

When a folder is being marked as shared by A, the adversary shall save a copy of the private version of the folder before node key re-encryption. When A sends the folder sharing request to B, the adversary can retrieve the sent share key via the MITM attack.

Now the server has access to the share key, and with it, the node keys of each of the files and subfolders. In the case that the folder was populated before being made shareable, some of the files that were in the folder before being shared might have remained until after the share key was given to B.

Let's define that there are N nodes in the pre-shared version of the folder, and M nodes in the post-shared version. Which of the pre-share nodes match which of the post-share nodes can be found by using all of the post-share node keys to decrypt the headers of pre-share nodes, and seeing which key retrieves properly formatted data for which header. Because some files may have been removed or added in the time between the folder being marked shareable and the share-key being sent, not all pre-share nodes may have matching keys.

After the matching has been done, the adversary will have the node key of matched nodes in the pre-share version of the folder. The adversary will also have access to the nonces and meta-macs for each of the files. The adversary knows for each matched file

the obfuscated file key, and the encryption of it under the victim's master key, leading to $2 \cdot J$ AES-ECB plaintext/ciphertext pairs recovered from J files successfully matched.

6.4.3 Complexity. This attack has $O(N \cdot M)$ worst-case time complexity, for N and M being the total numberof files in the original private folder and shared folder respectively.

6.4.4 Consequences. With this attack, an adversary is able to recover data from a previously private version of a secure folder. The adversary will be able to determine that matched files were in the folder before the folder was made shareable. In the event that some of the files were edited after the folder was made shareable, but before the share key was intercepted, the adversary will also be able to decrypt an older version of those files, as long as the node key was not changed.

This attack also recovers AES-ECB plaintext/ciphertext pairs under the victim's masterkey, which can then be used to perform other integrity attacks on the user's keys and data.

6.4.5 Limitations. The amount of matched files J depends on how many changes the user makes in the shared folder before sending a share key to a contact that has been MITM'd by the adversary. Any files that were removed before the share key was sent, cannot be matched.

6.5 Shared Folder Invitation Encryption Oracle

6.5.1 Threat Model. This attack assumes an adversary that has taken control of MEGA's servers, and that has an active MITM attack ongoing between the involved users A and B. The goal of the adversary is to retrieve an encryption of an arbitrary 128-bit plaintext block of its choosing under user A's master key.

6.5.2 Attack Description. When a contact B sends a share folder request to user A, and that request is accepted, then A receives the 128 bit AES Share Key k_S for the folder from B, encrypted using A's public RSA key. After receiving, decrypting and verifying k_S, it is re-encrypted using A's k_M into $[k_S]_{k_M}$ and sent to the server for storage.

When nodes in a shared folder are re-encrypted using k_S, this involves re-encrypting the obfuscated node key using k_S, and storing $[k_{obf}]_{k_S}$ next to the encryption under B's k_M, $[k_{obf}]_{k_M^B}$.

An adversary with an active MITM attack ongoing, could read and overwrite the k_S that A receives, to be whatever 128-bit data block the adversary wants an encryption for, k_S^*. The adversary would then also need to re-encrypt the obfuscated node keys of the folder using k_S^*, into being $[k_{obf}]_{k_S^*}$. Upon receiving k_S^*, A will verify the contents of the folder, because the node key encryptions A receives were done under k_S^*. Upon accepting k_S^*, an encryption $[k_S^*]_{k_M^A}$ will be sent to the adversary.

Users A and B can see the different obfuscated node key encryptions attached to the nodes in the shared folder. In order for the attack to remain covert, the server must continually send all node keys $[k_{obf}]_{k_S}$ to B and $[k_{obf}]_{k_S^*}$ to A whenever queried. If either user sees the other's share key encryptions, then they would detect that the shared folder nodes have been re-encrypted more times than B intended.

The sequence of actions in the attack can be seen in figure 6.

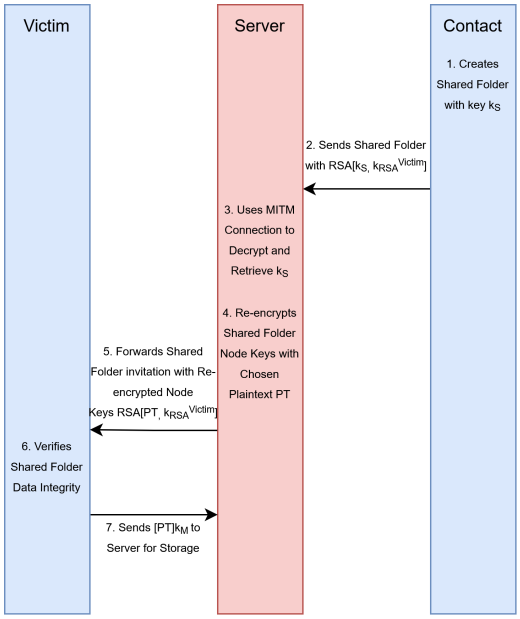

Figure 6: Sequence of actions taken by Victim, Contact and Server in Shared Folder Invitation Encryption Oracle attack.

6.5.3 Complexity. This attack takes O(1) time to complete for a given user.

6.5.4 Consequences. The presence of an arbitrary plaintext encryption oracle allows an adversary to perform many further attacks on a user. A short example, is that with the possession of $[0^{128}]_{k_M}$ and a pair $[N]_{k_M}||[N+1]_{k_M}$ for some N unique from the rest of the contacts, an adversary would be able to forge an AES-GCM contact fingerprint encryption. The attacker would eventually be able to forge encryptions for every aspect of the system encrypted under the user's k_M.

6.5.5 Limitations. If A and B manually compare the share keys that they use for decrypting the shared folder, this attack will be detected. The victim of this attack can also detect the attack on their own, if they see that the share keys that they receive contain very specific or suspicious patterns, which would enable attacks elsewhere. The blocks 0^{128} and $0||1^{32}||0^{96}$ from above could be considered suspicious.

This attack can only be performed covertly when A or B sends a legitimate share folder request to the other. This limits the number of invocations of this encryption oracle to the user. It is also possible to create a fake share folder request and send this to one or the other, but then the attack will be detected as soon the contact is asked about the folder request they received. This would also reveal that an active MITM attack is taking place.

7 MITIGATIONS

In order to mitigate the attacks described here, we have a small number of suggestions. We suggest two smaller and more immediately implementable fixes, namely:

- #1: Changing AES-GCM nonce byte length.
- #2: Encrypt Asymmetric Private Keys with AES-GCM.

However, as these are simple countermeasures, they do not solve the underlying problems allowing for these attacks. For that, we also suggest two more fundamental mitigations, namely:

- #3: Generate Multiple Master Keys for Strict Key Separation.
- #4: Replace all usage of AES-ECB with AES-GCM.

7.1 Short-Term Countermeasures

In the first attack, the recovery of Plaintext/Ciphertext pairs is possible only with the recovery of J_0, which is only possible for an adversary when the underlying Nonce is 12 bytes long. This leads to the suggestion for patch #1: Changing AES-GCM nonce byte length. By changing the length of the nonce used in the application's AES-GCM encryption, the attacker would need the plaintext/ciphertext pair $0^{128}/[0^{128}]_{k_M}$ in order to recover J_0 via the GHASH step. This would prevent the attack as described.

In the second attack, the lack of integrity protection on private user asymmetric keys is what allows for the Ed25519 private key to be replaced. This leads to the suggestion for patch #2: Encrypt Asymmetric Private Keys with AES-GCM. By changing the encryption scheme from AES-ECB to AES-GCM, the replacement of the key would also require the adversary to generate a valid GHASH tag. Without knowledge of the subkey plaintext/ciphertext pair $0^{128}/[0^{128}]_{k_M}$, this would not be possible.

Applying these two fixes would prevent the attack sequence leading to attacks 3 and 4.

7.2 Long-Term Countermeasures

These attacks are possible due to flaws in the fundamental cryptographic infrastructure of MEGA's cloud storage, namely that the architecture makes heavy use of AES-ECB to encrypt without integrity protection, and does not employ strict key separation for each of the sections where the user master key is used. The short-term patches suggested above, prevent the attacks described here, but the vulnerabilities associated would likely enable other attacks that are not described in this paper. Here we describe mitigations that would prevent these attacks as well as other yet undiscovered ones, but would be more costly to implement.

In order to prevent the last attack, we suggest to replace the system for encrypting and re-encrypting nodes in storage to make use of an implementation of Updateable Encryption [8].

We also further echo suggestions previously mentioned in [7], and to implement strict key separation as patch #3, and to implement AES-GCM encryption for all encryption of keys as patch #4. The use of AES-GCM or similar integrity assuring encryption across MEGA's infrastructure means integrity protection for keys, by forcing adversaries to also generate valid cryptographic tags for any ciphertexts they want to replace. Simply having a known plaintext/ciphertext would no longer be enough to forge an encryption.

For strict key separation, the usage of a singular master key to encrypt both contact fingerprints, private asymmetric keys, and storage node keys, is what enables attacks that depend on interactions between these disparate parts of MEGA's services. By enforcing that unrelated parts of the secure storage must use separate keys for encryption, these interactions would not occur, and would lead to fewer vulnerabilities.

A way to do this while minimizing the amount of re-encryption, is for each user to generate three more master keys B, C and D. The original master key A will remain, but will only be used for secure storage node key encryption. Meanwhile, key B will used for encrypting the user's asymmetric private keys,key C will be used to encrypt fingerprints of the user's contacts, and D to encrypt incoming shared folder share keys. This minimizes the amount of re-encryption that would need to be performed down to only the asymmetric keys, contacts and share keys. With this patch in place, attacks could not be performed that require interaction between these three disparate sections of the architecture, solving any other undiscovered attacks that would have made use of this.

8 EXPERIMENTS

In order to validate these attacks, a Proof of Concept for them was developed that simulates MEGA's systems and services. The Shared Folder Node Key Private Encryption Matching attack was then tested in an experiment using the Full Plaintext Wikipedia Dataset[4], in order to verify the volume and percentage of plaintext/ciphertext pairs it successfully matches. Proposals for mitigations against the attacks were also then tested, by implementing them as separate patches and testing whether an attack is successful in spite of them.

8.1 Proof of Concept

These attacks were implemented in a Proof of Concept simulation of MEGA's systems. This simulation was implemented using Python, and was based off of the technical specification for MEGA's architecture in the MEGA 2022 June Security Whitepaper [12], as well as the open-source MEGA Client Application SDK [3]. The proof of concept does not interact with MEGA Applications, nor with MEGA's servers, nor does it involve in its code any of the source code from the MEGA Client Application SDK.

The Proof of Concept simulates two types of entities, a User and a Server. The User object has methods for performing requests to the server, simulating actions that an actual user would perform when using MEGA's services. The User also has several methods for attempting to verify the data it receives from the server, both implementing tests that MEGA's applications automatically run, and other potential tests that could be done manually by a user with the raw data received.

The Server object simulates MEGA's server side. As MEGA's client side code is not publicly available, the Server object was implemented with the goal of successfully interfacing with the User object's requests in a way that conforms to the publicly available specification.

These constraints thus require the attacks to be implemented in a way, where the results that the Server object gives back to the User object must not contain any detectable faults, as far as is possible to detect given the security architecture. All the attacks implemented as described in the attacks section were successful in retrieving plaintext data and making changes to storage that should not have been possible for the Server given the security model claimed by MEGA, and that the User was not able to detect.

A link to the source code of the proof of concept can be found in appendix A.

Number of Nodes in Folder	Number of Matches	% of Matches
11696	11696	100%

Table 1: Success rate of Shared Folder Node Key Private Encryption Matching attack on Plaintext Wikipedia Dataset

8.2 Matching Attack Success Rate Measurement

8.2.1 Methodology. The experiment is performed on the 4th attack, namely the Shared Folder Node Key Private Encryption Matching attack. It was performed using the Full Plaintext Wikipedia Dataset, sourced from [4]. The entire dataset is turned into a ProtFolder object, and then re-encrypted to turn it into a shared folder. From the shared folder, the number of nodes is recorded, and no changes are made to the dataset before the shared folder invitation is sent. After the matching attack is performed, the number of successful matches is then recorded, and the percentage of successful matches compared to the number of nodes in the shared folder is calculated.

8.2.2 Results. The results of the experiment can be found in table 1. Of the 11696 nodes in the dataset, all node keys were successfully matched, leading to a matching success rate of 100%.

8.3 Testing Proposed Mitigation Strategies

In section 7, mitigation strategies are proposed against the attacks described in section 6, as methods to prevent the full attack chain from being performed. These mitigation strategies were:

- Patch #1: Changing AES-GCM nonce byte length to 13.
- Patch #2: Encrypt Asymmetric Private Keys with AES-GCM.
- Patch #3: Generate Multiple Master Keys for Strict Key Separation.
- Patch #4: Replace all usage of AES-ECB with AES-GCM.

In order to validate the effectiveness of these patches, this experiment implements them into the Proof of Concept, to test them against the 5 proposed attacks:

- Attack 6.1: Encrypted Contact Fingerprint Plaintext/Ciphertext Pair Recovery
- Attack 6.2: Ed25519 Signature Key Replacement
- Attack 6.3: First-Contact Establishment Covert Ed25519 Key Replacement
- Attack 6.4: Shared Folder Node Key Private Encryption Matching
- Attack 6.5: Shared Folder Invitation Encryption Oracle

The mitigation strategies were developed as modifications to the User simulation object, with each mitigation strategy developed as a separate User simulation object. A User object was also created with the application of all 4 patches at once. Any relevant functions implemented in the User object, are modified to adhere to the rule described in the mitigation strategy. This resulted in five different patching strategies.

Patch #3 was implemented by separating the User's k_M into several separate keys, in order to ensure that blocks of plaintext data in these four disparate sections of the architecture are encrypted with different keys:

- $k_M^{Storage}$ for encrypting user data obfuscated node keys
- $k_M^{PrivateKeys}$ for encrypting user asymmetric communication private keys
- $k_M^{Contacts}$ for encrypting user contact signature key fingerprints
- $k_M^{SharedFolders}$ for encrypting received shared folder k_S keys

Each attack was tested in an environment with each patched version of the User object, in order to track which attacks remained effective after the patches. Attacks were tested in two versions: one where the attack is attempted as originally described, and another where the prior knowledge of an attacker is modified to attempt to circumvent the attack. These modifications exist to attempt a weaker version of the attack, where either the goal of the attack is less severe or where the attacker is required to be much more powerful to perform it. Attacks that were modified in such a way are denoted separately from the original attack, as they are a different attack in nature. These are described in more detail as follows.

8.3.1 Modified Attack 6.1 Differences. For patch #3, the attacker's goal has changed to retrieve a plaintext/ciphertext pair that has been encrypted under the user's master Key. This makes the attack weaker, with the PT/CT pair retrieved by the attack only being applicable to the section of the security architecture encrypted under $k_M^{Contacts}$.

8.3.2 Modified Attack 6.2 Differences. To account for Patches #2 and #4, the attacker is assumed to also possess an AES-GCM random plaintext/ciphertext pair oracle. This makes the attack weaker, requiring a stronger attacker that has both plaintext/ciphertext pair oracles in order to be able to perform the attack.

8.3.3 Modified Attack 6.3 Differences. No modified version of the attack is proposed.

8.3.4 Modified Attack 6.4 Differences. For Patch #3, the attacker's goal becomes collecting plaintext/ciphertext pairs encrypted under $k_M^{Storage}$. As each node key in storage should be unique, this narrows the use cases for these pairs.

For the case of Patch #4, the attacker's goal has shifted to collecting AES-GCM encrypted plaintext/ciphertext pairs. As these are encrypted using nonces whose uniqueness is enforced by the client, the use cases for these pairs are much narrower.

8.3.5 Modified Attack 6.5 Differences. For Patch #3, the goal of the attack has shifted to encrypting an attacker's chosen plaintext under $k_M^{SharedFolders}$. This narrows the sections of the security architecture where the encryption can be used.

For Patch #4, the goal of the attack has shifted to encrypting an attacker's chosen plaintext under AES-GCM. This narrows the use cases for the encryptions, as AES-ECB encryptions could have been used to generate valid encryptions under other schemes, such as Counteer Mode or Cypher Block Chaining, but this is no longer possible with AES-GCM encrypted ciphertexts.

8.3.6 Results. Table 2 describes the results. In the table it can be seen that the most effective mitigation strategy is to apply all of the suggested fixes together, as it blocks the most attack variants. However, even with all of the patches applied, weaker variants of the attacks are still possible to perform. These attacks do result in an attacker gaining knowledge it should not have access to, but the

Attack	Patch #1	Patch #2	Patch #3	Patch #4	All Patches
Attack 6.1	✓	X	✓	X	✓
Attack 6.1*	✓	X	X	X	✓
Attack 6.2	X	✓	✓	✓	✓
Attack 6.2*	X	X	X	X	X
Attack 6.3	✓	✓	✓	✓	✓
Attack 6.4	X	X	✓	✓	✓
Attack 6.4*	X	X	X	X	X
Attack 6.5	X	X	✓	✓	✓
Attack 6.5*	X	X	X	X	X

Table 2: Table of Patches and Attacks, showing which attacks were prevented (✓) and which attacks succeeded when presented with the patch (X). Attacks with a (*) have been modified in their assumption about the attacker to attempt to circumvent the patch. The modifications are described in subsection 8.3.

.

patches mean that the knowledge gained is much less versatile and cannot be used in some of the other proposed attacks.

9 DISCUSSION

One of the main limitations in the proof of concept for the attacks, is that they were not implemented on a client-side application for MEGA's services. The development of such attacks would require the use of a proxy connection to MEGA's servers, which was successfully implemented in the proof of concept used in the research by Backendal et.al. [7], but would require the coordination of two separate client programs connecting to the same proxy and the proxy managing them appropriately, adding complexity to its development.

Due to MEGA's reliance on AES-ECB for much of the encryption work in their services' architecture, the presence of the plaintext/ciphertext pairs found via these attacks likely also enable other attacks on user's secure data. This means that even in the case that MEGA patches these attacks, future work has value in researching further attacks that assume an arbitrary plaintext/ciphertext pair oracle for the adversary.

10 CONCLUSION

In spite of the patches that MEGA has applied to their systems to prevent attacks found in previous research and bug reports, there are still several simple vulnerabilities in their system. The vulnerabilities enable a potentially adversarial cloud provider to read and edit a user's storage and contacts without their knowledge. This undermines the promise of MEGA's zero-knowledge encryption and that even meddling by MEGA should be impossible.

We echo the statement by Backendal et. al. in their research, that research into developing a standardized system for secure cloud storage should be developed as a means to eventually prevent these kinds of security issues in the future [7]. A peer reviewed system developed in collaboration with researchers and stakeholders provides a more rigorous context where more eyes are able to detect vulnerabilities before it is implemented in practice, while the stakeholders can make it clear and ensure that the system provides the features that they and their clients want from it.

REFERENCES

[1] 2025. About us. (Sept. 2025). https://mega.io/about

[2] 2025. MEGA: Protect your Online Privacy. (Sept. 2025). https://mega.io/

[3] 2025. meganz/sdk. (Sept. 2025). https://github.com/meganz/sdk original-date: 2013-11-20T12:33:53Z.

[4] 2025. Plaintext Wikipedia (full English). (Sept. 2025). https://www.kaggle.com/datasets/ffatty/plaintext-wikipedia-full-english

[5] 2025. Security and privacy. (Sept. 2025). https://mega.io/security

[6] Martin R. Albrecht, Miro Haller, Lenka Mareková, and Kenneth G. Paterson. 2023. Caveat Implementor! Key Recovery Attacks on MEGA. In *Advances in Cryptology – EUROCRYPT 2023*, Carmit Hazay and Martijn Stam (Eds.). Springer Nature Switzerland, Cham, 190–218. https://doi.org/10.1007/978-3-031-30589-4_7

[7] Matilda Backendal, Miro Haller, and Kenneth G. Paterson. 2023. MEGA: Malleable Encryption Goes Awry. In *2023 IEEE Symposium on Security and Privacy (SP)*. 146–163. https://doi.org/10.1109/SP46215.2023.10179290 ISSN: 2375-1207.

[8] Colin Boyd, Gareth T. Davies, Kristian Gjøsteen, and Yao Jiang. 2020. Fast and Secure Updatable Encryption. In *Advances in Cryptology – CRYPTO 2020*, Daniele Micciancio and Thomas Ristenpart (Eds.). Springer International Publishing, Cham, 464–493. https://doi.org/10.1007/978-3-030-56784-2_16

[9] Dataintelo. 2025. Consumer Cloud Storage Services Market Report | Global Forecast From 2025 To 2033. (Sept. 2025). https://dataintelo.com/report/consumer-cloud-storage-services-market

[10] Nadia Heninger and Keegan Ryan. 2023. The Hidden Number Problem with Small Unknown Multipliers: Cryptanalyzing MEGA in Six Queries and Other Applications. In *Public-Key Cryptography – PKC 2023*, Alexandra Boldyreva and Vladimir Kolesnikov (Eds.). Springer Nature Switzerland, Cham, 147–176. https://doi.org/10.1007/978-3-031-31368-4_6

[11] David A McGrew, John Viega, and San Jose. 2008. The Galois/Counter Mode of Operation (GCM). (Aug. 2008).

[12] MEGA. 2022. *MEGA Security Whitepaper - Third Edition.* Technical Report. MEGA, Auckland, New Zealand. https://mega.nz/SecurityWhitepaper.pdf

[13] Naoki Shimoe and Noboru Kunihiro. 2025. Key Recovery Attacks on Unpatched MEGA from Four Queries: Solving Approximate Divisor Problem with Help of Approximation of Squared Divisor. In *Applied Cryptography and Network Security*, Marc Fischlin and Veelasha Moonsamy (Eds.). Springer Nature Switzerland, Cham, 547–571. https://doi.org/10.1007/978-3-031-95761-1_19

A PROOF OF CONCEPT CODEBASE

https://github.com/K-J-Kiisa/mega-sec