Cryptanalyzing MEGA in Six Queries

Keegan Ryan and Nadia Heninger

University of California, San Diego
kryan@eng.ucsd.edu,nadiah@cs.ucsd.edu

Abstract. In recent work, Backendal, Haller, and Paterson identified several exploitable vulnerabilities in the cloud storage provider MEGA. They demonstrated an RSA key recovery attack in which a malicious server can recover the client's RSA private key. Their attack uses binary search to recover the private RSA key after 1023 client logins, and optionally could be combined with lattice methods for factoring with partial knowledge to reduce the number of logins to 512 in theory, or 683 in the published proof of concept.
In this note, we give an improved attack that requires only six client logins to recover the secret key. Our optimized attack combines several techniques, including a modification of the extended hidden number problem and the structure of RSA keys, to exploit additional information revealed by MEGA's protocol vulnerabilities. MEGA has emphasized that users who had logged in more than 512 times could have been exposed; these improved attacks show that this bound was conservative, and that unpatched clients should be considered vulnerable under a much more realistic attack scenario.

1 Introduction

MEGA is an encrypted cloud storage provider whose protocols are designed to protect a client's data and secret key against a malicious server or malicious entity in the backend infrastructure. In a recent paper [1], Backendal, Haller, and Paterson detail multiple exploitable flaws in MEGA's protocols including a full key recovery attack [1, Section III]. In this attack, a malicious server sends mauled ciphertexts to a victim client as the client logs in to a user's account. The client processes this data using their own secret information and leaks partial information about the user's private key in their protocol response to the server. Backendal, Haller, and Paterson detail how the malicious server can use binary search to recover the private key after 1023 client logins, or use standard lattice methods to reduce the number of logins to 512.

MEGA has patched the issue by adding additional payload validation and emphasized in their blog post about the vulnerabilities [6] that only clients who have logged in more than 512 times are vulnerable.

We give an improved attack which operates under the exact same scenario but only requires six login attempts from a victim client. Our attack exploits the interplay between the symmetric and asymmetric cryptographic operations in MEGA's design. The attack recovers the data in several distinct stages, and it illustrates several techniques for recovering sensitive key material in a real-world scenario.

The patches that MEGA developed to mitigate the original key recovery attack are effective against our improved attack as well, so updated clients are not vulnerable to the techniques presented in this work. However, our optimized cryptanalysis underscores the ongoing risk to unpatched clients.

Overall, these attacks demonstrate how simple errors in cryptographic design can lead to unexpectedly devastating attacks.

1.1 Attack Overview

We present two novel and overlapping attacks. The first (*fast*) attack requires only 17 login attempts on average and is quite computationally inexpensive. The second (*small*) attack requires only 6 login attempts to succeed with 98% probability, but it is more computationally intensive. This latter attack can be performed in 4.5 hours on an 88-core machine, but we include both because the former can be easily verified and includes some interesting additional analysis steps. Both of these attacks proceed in roughly the same series of stages with only minor variations in how the stage is completed in both the fast attack and the small attack.

In MEGA's login protocol, the server sends the client an RSA private key that is encrypted using AES in ECB mode. The client decrypts the RSA private key, uses this RSA private key to decrypt a session ID that the server has encrypted to the RSA public key, and sends the result to the server.

The attack of Backendal, Haller, and Paterson modifies the ECB-encrypted ciphertext of the RSA private key and the encrypted session ID to obtain one bit of information about the secret key per login. However, the client is using the modified secret key to send 43 contiguous bytes of information from the result of the RSA decryption to the server. In our attack, the adversary swaps blocks in the ECB-encrypted wrapped RSA key before sending it to the client and then analyzes the resulting message from the client to obtain more information about the RSA secret key per victim client login attempt.

In the first stage of analysis, the attacker represents the 43-byte leakage from the client in terms of the unknown AES plaintext blocks. Second, these algebraic representations are manipulated so that the attacker learns information about the most significant bytes (MSBs) of an unknown value, not just about a contiguous subsequence of bytes. In the fast attack, this is done using an approach from solutions to the Extended Hidden Number Problem [4], and in the small attack, this is done by brute forcing unknown most significant bytes. Third, in the fast attack, these approximations of the MSBs are refined by combining approximations together so more MSBs are known. This is why the fast attack requires more samples than the small attack. Fourth, the (refined) approximations are used to solve for the value of multipliers in the algebraic representation. These unknown multipliers correspond to differences between plaintext blocks in the encoded RSA private key. Fifth, we use the RSA equations to brute force a block of plaintext bytes of the RSA private exponent in the encoded key, and the plaintext differences reveal the values of other plaintext blocks. Finally, the plaintext blocks containing the MSBs of one of the RSA factors are analyzed in a Coppersmith attack [2] to recover the full factorization and private key.

Section 2 recalls the full details of the attack context and the original cryptanalysis. Sections 3 through 8 discuss each of the stages in turn. Section 9 analyzes the overall complexity of the attack, and Section 10 describes experiments.

2 Background

2.1 Notation

Our notation essentially follows the notation used in [1]. Lists and arrays are 1-indexed, and pt_i (and ct_i) refer to the i^{th} 16-byte block of plaintext (ciphertext) in the AES-encrypted encoding of the RSA private key. $\mathrm{E}_{\mathrm{AES}}$ and $\mathrm{D}_{\mathrm{AES}}$ are AES encryption and decryption using a user's master AES key. Concatenation is denoted by $\|$. Reduction modulo N may be used to define a congruence, the operation of taking an equivalent value in $\{0, \ldots, N-1\}$, or the operation of taking an equivalent value in $\{-\lfloor (N-1)/2 \rfloor, \ldots, \lfloor N/2 \rfloor\}$, depending on context.

2.2 Attack Context

When a MEGA user attempts to log in for the first time on a new client, the client is only in possession of an AES secret key derived from the user's password. To function properly, the client requires a copy of the user's RSA private key. The server possesses the user's RSA public key and a copy of the user's RSA private key encrypted using the AES key in ECB mode, so in theory the private key is hidden from the server, but the client can obtain the private key by decrypting the encrypted private key from the server with the AES secret key. It is the malicious server's goal is to recover the user's private key.

During the login process, the server creates a 43-byte session identifier (SID), which it encrypts using the RSA public key and sends to the client alongside the wrapped private key. The client uses the AES key to unwrap the RSA private key, then uses the parameters in the unwrapped key to decrypt the RSA ciphertext and retrieve the SID. The client then sends the retrieved SID to the server. The malicious server wishes to use the SID value sent from the client to infer information about the parameters in the unwrapped private key.

Several of the exact implementation details are relevant for our improved attack, so we recount them here. The remaining details can be found in Backendal, Haller, and Paterson's paper [1, Section II]. We denote the RSA public key by (N, e), where the factors of modulus N are p and q. The public RSA exponent is e and the private exponent is d. The public RSA exponent is set by the client; the web client[1] uses $e = 257$, and the SDK[2] uses $e = 17$. MEGA clients use RSA-CRT for decryption, so let $u \leftarrow q^{-1} \bmod p$ be the coefficient used during CRT operations.

The private key is encoded as

$$sk_{share}^{encoded} \leftarrow \mathrm{l}(q) \parallel q \parallel \mathrm{l}(p) \parallel p \parallel \mathrm{l}(d) \parallel d \parallel \mathrm{l}(u) \parallel u \parallel P.$$

l encodes the bit length of different values as a 2-byte integer, all integers are stored in big-endian format, and P is an 8-byte padding value unknown to the adversary. We wish to highlight that q is 1024 bits in length, so $\mathrm{l}(q)$ is `0x0400`, and since the secret values are of

[1] `https://github.com/meganz/webclient/blob/9fca1d0b7d8a65b9d483a10e798f1f67d1fb8f1e/js/crypto.js#L207`

[2] `https://github.com/meganz/sdk/blob/849aea4d49e2bf24e06d1a3451823e02abd76f39/src/crypto/cryptopp.cpp#L798`

predictable size, they appear at predictable offsets within the plaintext. We also highlight that the private key encodes the full private exponent d and does not include the private exponents d_p, d_q that are frequently stored for use in RSA-CRT decryption. Finally, we note that due to the length fields, the 1024-bit u value spans 9 AES plaintext blocks, and the first and last of those contain the length and padding fields respectively. As in the original attack, we constrain our attacker to not alter these length and padding fields.

This encoding of the private key is 656 bytes, or 41 AES blocks. The encoded private key is encrypted using AES in ECB mode, which means that each 16-byte plaintext block is encrypted independently. That is,

$$ct_1 \parallel ct_2 \parallel \cdots \parallel ct_{41} = \mathrm{E}_{\mathrm{AES}}(pt_1) \parallel \mathrm{E}_{\mathrm{AES}}(pt_2) \parallel \cdots \parallel \mathrm{E}_{\mathrm{AES}}(pt_{41}).$$

Decryption of the encrypted private key also processes 16-byte blocks independently, enabling malleability attacks where the malicious server alters individual blocks of ciphertext to alter the corresponding blocks of plaintext in the private key encoding.

When the server constructs the RSA plaintext with the 43-byte SID, it typically places the SID in bytes 3-45 of the 256-byte RSA plaintext m. Prior to patching, clients extract these bytes from the RSA decryption output without checking the validity of the remainder of the decryption output. However, there is special behavior in the extraction function that checks if byte 2 is nonzero, and if this is the case it extracts bytes 2-44. This detail has no consequence for the RSA key extraction attack in [1], but it is a necessary aspect of our small attack. If we assume the output bytes of the RSA decryption function are uniformly distributed, clients have probability 255/256 of returning $SID \leftarrow m[2:44]$. We temporarily set this detail aside and assume that all SIDs returned by the client are composed of these bytes, and we revisit it in Section 9.

MEGA clients use Garner's formula [3] to perform RSA-CRT decryption, the process of decrypting an RSA ciphertext c to message m. These equations, as well as the SID extraction step, are detailed below.

$$m_p \leftarrow c^{d \bmod (p-1)} \bmod p$$

$$m_q \leftarrow c^{d \bmod (q-1)} \bmod q$$

$$m \leftarrow ((m_p - m_q)u \bmod p)q + m_q$$

$$SID \leftarrow m[2:44]$$

2.3 Original Attack of Backendal, Haller, and Paterson

In the original attack, the adversary alters ciphertext block ct_{40}, which is the last ciphertext block corresponding to only bytes of u, and no length fields or padding bytes. The attacker sends this altered wrapped key and RSA ciphertext $q_{guess}^e \bmod N$ to the client. The client decrypts and decodes the wrapped key to obtain private key (q, p, d, u', P) where $u' \neq u$ is not the correct value of $q^{-1} \bmod p$ to use during RSA-CRT decryption.

If $q_{guess} < q$, then $m_p \equiv m_q \pmod p$, so $h = 0$ and $m = m_q < q$. Thus all SID bytes are 0. If $q_{guess} >= q$, then $m_p \not\equiv m_q \pmod p$, so $h \neq 0$ and $m > q$. Thus the SID bytes are

nonzero with high probability. The attack therefore uses whether SID is zero or nonzero as an oracle for whether the attacker-chosen q_{guess} is smaller or larger than the secret q. The adversary performs a binary search on the value of q until sufficiently many most significant bits of q are known.

The attacker then uses a cryptanalytic technique by Coppersmith [2] to recover the least significant bits of q, and thus obtain the full factorization of N. Asymptotically, this attack recovers the factorization of N in polynomial time once the attacker knows the most significant half of bits of q. In the context of MEGA, that is 512 bits, which requires 512 login attempts to obtain. In practice, this attack can be prohibitively slow at the asymptotic limit, and implementations of Coppersmith's method often use additional most significant bits, which makes the implementation faster and more understandable. The proof-of-concept code associated with the original attack uses 683 most significant bits and therefore requires 683 login attempts.

We observe that although the client provides the adversary with 344 bits of SID per login attempt, this original attack only uses this data to refine the knowledge of the private key by a single bit. It is therefore natural to wonder if the client's responses can be exploited in a more sample-efficient way, recovering the same private key with fewer login attempts. This is what our new attacks accomplish.

3 Expressing Leakage Algebraically

We begin our cryptanalysis by demonstrating how to use the information returned during a login attempt to create some algebraic expression. As in the original attack, the adversary alters ciphertext blocks corresponding to the value of u, and therefore the client uses the altered u' value when performing decryption, but the remaining private key values are unaltered. In our attack, the adversary also picks an RSA ciphertext c at random, and reuses the same c for each login attempt. Both the (modified) wrapped key and RSA ciphertext are sent to the client during a login attempt.

By combining Garner's formula for RSA decryption with the extraction of the SID s' with altered value u', this gives the congruence

$$(m_p - m_q)u'q + m_q \equiv e'_1 2^{b_1} + s'2^{b_2} + 2^{b_2-1} + e'_2 \pmod{N}.$$

The left hand side expresses the output of the decryption function in terms of its input, and the right hand side expresses the output in terms of the known SID bytes $s' = m'[2:44]$ and the other unknown bytes $e'_1 = m'[1]$ and $e'_2 = m'[45:256] - 2^{b_2-1}$. The 2^{b_2-1} term is present so that unknown e'_2 may be positive or negative and so $|e'_2|$ is minimized.

We can construct a similar equation using altered value u'' and SID s''.

$$(m_p - m_q)u''q + m_q \equiv e''_1 2^{b_1} + s''2^{b_2} + 2^{b_2-1} + e''_2 \pmod{N}.$$

Subtracting these two congruences, we have

$$(u' - u'')(m_p - m_q)q \equiv (e'_1 - e''_1)2^{b_1} + (s' - s'')2^{b_2} + (e'_2 - e''_2) \pmod{N}.$$

The adversary can give extra structure to $(u' - u'')$ by carefully manipulating the AES-encrypted key. The value u used by the client during RSA decryption is decoded from the 9 plaintext blocks $\mathrm{D}_{\mathrm{AES}}(ct_{33} \parallel ct_{34} \parallel \cdots \parallel ct_{41})$. Plaintext blocks pt_{33} and pt_{41} also include some bytes of d, the encoding of $\mathrm{l}(u)$, and padding P. Now observe that if the attacker swaps out some of these ciphertext blocks encrypting u with ciphertext blocks ct_i, ct_j of their choosing, the decrypted and decoded value of u used by the client will contain bits from pt_i and pt_j. Consider what happens when the client decodes u' and u'' from the following two ciphertexts:

$$u' = \mathrm{Decode}[\mathrm{D}_{\mathrm{AES}}(ct_{33}) \parallel \mathrm{D}_{\mathrm{AES}}(ct_i) \parallel \cdots \parallel \mathrm{D}_{\mathrm{AES}}(ct_i) \parallel \mathrm{D}_{\mathrm{AES}}(ct_i) \parallel \mathrm{D}_{\mathrm{AES}}(ct_{41})]$$
$$u'' = \mathrm{Decode}[\mathrm{D}_{\mathrm{AES}}(ct_{33}) \parallel \mathrm{D}_{\mathrm{AES}}(ct_i) \parallel \cdots \parallel \mathrm{D}_{\mathrm{AES}}(ct_i) \parallel \mathrm{D}_{\mathrm{AES}}(ct_j) \parallel \mathrm{D}_{\mathrm{AES}}(ct_{41})],$$

After decryption, all of the plaintext blocks that contain only bits of u are replaced with pt_i, except for one in the second plaintext which is replaced with pt_j. The plaintext blocks which contain length encoding data or padding are not modified, so validation of the plaintext succeeds. With this construction, $(u' - u'')$ has special structure, because the only difference between the two is in block 40, which corresponds to bytes 105 through 120 of the encoded u. Therefore, we have

$$u' - u'' = (pt_i - pt_j)2^{64}.$$

For simplicity, in the future we will denote $\delta_{i,j} = pt_i - pt_j$, and observe that $|\delta_{i,j}| < 2^{128}$.

We will also consider $u' - u'''$ when u''' was decoded from the ciphertext

$$u''' = \mathrm{Decode}[\mathrm{D}_{\mathrm{AES}}(ct_{33}) \parallel \mathrm{D}_{\mathrm{AES}}(ct_i) \parallel \cdots \parallel \mathrm{D}_{\mathrm{AES}}(ct_j) \parallel \mathrm{D}_{\mathrm{AES}}(ct_i) \parallel \mathrm{D}_{\mathrm{AES}}(ct_{41})]$$

that differs only in block 38. By the same logic as before,

$$u' - u''' = (pt_i - pt_j)2^{196} = 2^{128}\delta_{i,j}2^{64}.$$

This generalizes so that the adversary can construct values of u with difference $\delta_{i,j}2^{128t+64}$ for $t \in \{0, 1, \ldots, 5\}$.

4 Obtaining Most Significant Bytes

For any AES ciphertext block indices i and j, Section 3 gives us the capability to construct an equation involving the differences of the corresponding plaintexts $\delta_{i,j} = pt_i - pt_j$. Specifically, we have

$$\delta_{i,j}2^{128t+64}(m_p - m_q)q \equiv (e'_1 - e''_1)2^{b_1} + (s' - s'')2^{b_2} + (e'_2 - e''_2) \pmod{N}.$$

In this equation, the adversary knows $(s' - s'')$ because it is the difference of two SIDs, and the adversary also knows t, b_1, b_2, and N. The adversary does not know $2^{64}(m_p - m_q)q \bmod N$, but this value is constant throughout the attack. The adversary does not know $(e'_1 - e''_1)$ or $(e'_2 - e''_2)$, but knows they are bounded by $|e'_1 - e''_1| \leq E_1 = 2^8$ and $|e'_2 - e''_2| \leq E_2 = 2^{b_2}$.

The goal of this phase is to learn the most significant bytes of some algebraic expression. This is a generally useful goal because it allows us to represent the error in the approximation

as some bounded variable, and it is frequently possible to efficiently solve the problem of recovering bounded variables using lattice methods.

We now detail two approaches for obtaining the most significant bytes of this representation.

4.1 Brute Force

Because e'_1 and e''_1 are both single-byte values, $e'_1 - e''_1$ takes on one of 511 values. We can brute force these values and expect to eventually guess the correct value. Therefore, assuming we have guessed correctly, we can compute $a = (e'_1 - e''_1)2^{b_1} + (s' - s'')2^{b_2}$ and write

$$2^{128t}\delta_{i,j}x \equiv a + \varepsilon \pmod{N}$$

where $x = 2^{64}(m_p - m_q)q \bmod N$ is unknown but constant throughout the attack. $2^{128t}\delta_{i,j}$ is an unknown multiplier. ε is unknown and bounded by $|\varepsilon| \leq 2^{b_2} = 2^{1696}$.

4.2 Extended Hidden Number Problem

We observe that the problem of converting a sample with a known block of contiguous bytes into a sample with known most significant bytes (MSBs) resembles the Extended Hidden Number Problem (EHNP) [4], specifically the Hidden Number Problem with two holes (HNP-2H). To obtain the MSBs, we search for a known multiplier C which simultaneously makes the unknown terms $(e'_1 - e''_1)C2^{b_1} \bmod N$ and $(e'_2 - e''_2)C \bmod N$ small. If we assume $|e'_1 - e''_1| < E_1$ and $|e'_2 - e''_2| < E_2$, such a value of C can be found by reducing the lattice defined by the rows of the basis matrix $B =$

$$\begin{bmatrix} E_1 N & 0 \\ E_1 2^{b_1} & E_2 \end{bmatrix}.$$

Lattice reduction finds the shortest vector $v = (E_1(C2_1^b \bmod N), E_2 C)$ with $\|v\|_2 \leq \frac{2}{\sqrt{3}}\det \mathrm{B}^{1/2} = \frac{2}{\sqrt{3}}\sqrt{E_1 E_2 N}$. Thus

$$\begin{aligned} &|(e'_1 - e''_1)C2^{b_1} + (e'_2 - e''_2)C \bmod N| \\ \leq &|e'_1 - e''_1||C2^{b_1} \bmod N| + |e'_2 - e''_2||C| \\ \leq &E_1|C2^{b_1} \bmod N| + E_2|C| \\ \leq &\|v\|_2 + \|v\|_2 \\ \leq &\frac{4}{\sqrt{3}}\sqrt{E_1 E_2 N}. \end{aligned}$$

We set $C = v_2/E_2$ and note that C does not depend on information leaked from the client, and thus can be reused for every sample.

We therefore let $x = C(m_p - m_q)q \bmod N$, $a = C(s' - s'')2^{b_2}$, and $\varepsilon = (e'_1 - e''_1)C2^{b_1} + (e'_2 - e''_2)C \bmod N$. This yields

$$2^{128t}\delta_{i,j}x \equiv a + \varepsilon \pmod{N}.$$

$2^{128t}\delta_{i,j}$ is an unknown multiplier. x is unknown, but constant throughout the attack. a is known. ε is unknown and bounded by $|\varepsilon| \leq \frac{4}{\sqrt{3}}\sqrt{E_1 E_2 N} \leq 2^{1878}$.

The approach using the EHNP technique therefore produces a similar equation as the brute force approach, but the bound on the unknown ε is greater. In fact, this approach loses about half of the information exposed by the client; instead of knowing 43 MSBs, this transformation gives information about only 21.25 MSBs.

5 Refining Approximations

For later stages in the attack, it is necessary to improve the EHNP approximations. Specifically, for any AES block indices i, j and choice of $t \in \{0, 1, \ldots, 5\}$, the adversary uses Section 4 to learn a_t satisfying

$$2^{128t}\delta_{i,j}x \equiv a_t + \varepsilon_t \pmod{N}.$$

$\delta_{i,j} = pt_i - pt_j$ is the difference of two plaintexts and is bounded $|\delta_{i,j}| \leq 2^{128}$. We also have bound $|\varepsilon_t| < E$. The goal of the adversary is to *refine* the approximation by computing $\tilde{a}$ satisfying

$$\delta_{i,j}x \equiv \tilde{a} + \tilde{\varepsilon} \pmod{N}$$

where $|\tilde{\varepsilon}| \leq \tilde{E} \leq E$.

Since the new bound on the error is smaller, this is equivalent to learning additional MSBs of $\delta_{i,j}x$.

We simplify the problem to a single refinement step using two approximations. Once we show that this is possible, it is clear that this can be repeated multiple times to refine the approximation further. We state the problem generically.

Approximation Refinement Problem Assume the adversary is given $a_1, a_2, r \neq 0, N, E_1$ and E_2 satisfying

$$\begin{aligned} y &\equiv a_1 + \varepsilon_1 \pmod{N} \\ ry &\equiv a_2 + \varepsilon_2 \pmod{N} \\ |\varepsilon_1| &\leq E_1 \\ |\varepsilon_2| &\leq E_2 \\ 2|r|E_1 + 1 &\leq N - 2E_2. \end{aligned}$$

If $\min((2E_2 + 1)/|r|, 2E_1 + 1) < 2\tilde{E}$, then the attacker's goal is to return $\tilde{a}$ such that there exists $\tilde{\varepsilon}$ satisfying $|\tilde{\varepsilon}| \leq \tilde{E}$ and

$$y \equiv \tilde{a} + \tilde{\varepsilon} \pmod{N}.$$

To solve this problem, observe that there exists y satisfying $y \in [a_1 - E_1, a_1 + E_1]$. Without loss of generality, assume $r > 0$. so therefore $ry \in S_1 = [r(a_1 - E_1), r(a_1 + E_1)]$. Also observe that

$$ry \in S_2 = \bigcup_{k=-\infty}^{\infty} [a_2 - E_2 + kN, a_2 + E_2 + kN],$$

so we wish to find the intersection of S_1 and S_2. Because S_2 consists of the union of intervals of size $2E_2+1$, repeated at multiples of N, the gaps between these intervals are $N-2E_2-1$. Since the size of S_1 is $2rE_1+1 \leq N-2E_2-1+1$, S_1 intersects with at most one interval and we know there exists ry, the intersection of S_1 and S_2 is a single interval. Therefore

$$\begin{aligned} k^* &= \left\lceil \frac{r(a_1-E_1)-(a_2+E_2)}{N} \right\rceil \\ low &= \max(r(a_1-E_1), a_2-E_2+k^*N) \\ high &= \min(r(a_1+E_1), a_2+E_2+k^*N) \\ ry \in S_1 \cap S_2 &= [low, high] \\ \Rightarrow y &\in \left[\left\lceil \frac{low}{r} \right\rceil, \left\lfloor \frac{high}{r} \right\rfloor \right]. \end{aligned}$$

The size of this interval is at most $\min((2E_2+1)/r, 2E_1+1) < 2\tilde{E}$, so we let $\tilde{a}$ be its midpoint (or as close as possible if there are an even number of elements) and we have solved the problem.

To apply this to our specific problem, observe that this means that we can refine the EHNP sample $\delta i, jx \equiv a_0+\varepsilon_0 \pmod N$ with $2^{128}\delta_{i,j}x \equiv a_1+\varepsilon_1 \pmod N$ to quality $\tilde{E} = 2^{1750}$ because $r = 2^{128}$, $E_1 = E_2 = 2^{1878}$, $N \approx 2^{2048}$. Similar logic shows that we can use a_2 to refine even further, achieving a refined sample of the form

$$\delta_{i,j}x \equiv \tilde{a} + \tilde{\varepsilon} \pmod N \text{ with } |\tilde{\varepsilon}| \leq 2^{1622}.$$

This increases the number of MSBs known from about 21 to 53.

6 Recovering Unknown Multipliers

We now turn to the goal of recovering unknown and small multipliers. For arbitrarily many (i,j) pairs, the attacker knows $a_{i,j}$ such that

$$\delta_{i,j}x \equiv a_{i,j} + e_{i,j} \pmod N$$

where $|\delta_{i,j}| \leq 2^{128}$ and $|e_{i,j}| < E$. The value of E depends on if the adversary initially used the brute force strategy (giving $E = 2^{1696}$) in Section 4.1 or the EHNP strategy (4.2) plus refinement (5) (giving $E = 2^{1622}$).

Once again, we consider a generic form of this problem. A similar, although not identical, problem appears in [5], and our approach and lattice constructions are similar, although not identical.

Hidden Number Problem with Small Unknown Multipliers Given N, a_i, T, and E such that, $\forall 1 \leq i \leq d$,

$$\begin{aligned} t_i x &\equiv a_i + e_i \pmod N \\ |t_i| &\leq T \\ |e_i| &\leq E, \end{aligned}$$

the goal of the adversary is to recover all values of t_i.

Consider the case where $d = 2$. We show the following linear equation is small modulo N.

$$\begin{aligned} & t_2 a_1 - t_1 a_2 \\ \equiv & t_2(t_1 x - e_1) - t_1(t_2 x - e_2) && \pmod N \\ \equiv & -t_2 e_1 + t_1 e_2 && \pmod N \end{aligned}$$

Thus we have a linear system $ya_2 + za_1 \pmod N$ which takes on a small value $t_1 e_2 - t_2 e_1$ when evaluated at a small point $(t_2, -t_1)$. This lends itself well to a small vector in a particular lattice, spanned by the rows of the following basis:

$$B = \begin{bmatrix} 2E & 0 & a_1 \\ 0 & 2E & a_2 \\ 0 & 0 & N \end{bmatrix}$$

The vector $v = (2Et_2, -2Et_1, t_1 e_2 - t_2 e_1)$ is small and in this lattice, so we hope that lattice reduction finds such a short vector. However, note that if t_1 and t_2 share a common factor g, then $(2Et_2/g, -2Et_1/g, t_1 e_2/g - t_2 e_1/g)$ is also in the lattice, but it is even smaller. This complicates the analysis, but in practice lattice reduction finds this small vector (up to sign and division by a small factor) when $2ET \approx \|v\|_2 \lesssim \det(B)^{1/3} = (4E^2 N)^{1/3}$, or $T \lesssim (\frac{N}{2E})^{1/3}$.

Therefore, when this condition is satisfied, we learn $\pm t_1/\gcd(t_1, t_2)$ and $\mp t_2/\gcd(t_1, t_2)$. For the case $d > 2$, repeating this approach with t_i, $i \geq 2$ gives $\pm t_1/\gcd(t_1, t_i)$ and $\mp t_i/\gcd(t_1, t_i)$. In practice, computing the least common multiple of $\pm t_1/\gcd(t_1, t_2), \pm t_1/\gcd(t_1, t_3), \ldots, \pm t_1/\gcd(t_1, t_d)$ gives t_1 up to sign. This reveals the values of all other t_i up to the same sign. We omit a more fine grained analysis of the success probability. In practice, this suffices to solve the Hidden Number Problem with Small Unknown Multipliers.

We also briefly consider a generalized slightly larger lattice for the case $d = 3$. Consider the lattice spanned by the rows of basis

$$B' = \begin{bmatrix} 2E & 0 & 0 & a_1 \\ 0 & 2E & 0 & a_2 \\ 0 & 0 & 2E & a_3 \\ 0 & 0 & 0 & N \end{bmatrix}.$$

The vectors $v_1 = (2Et_2, -2Et_1, 0, t_1 e_2 - t_2 e_1)$ and $v_2 = (2Et_3, 0, -2Et_1, t_1 e_3 - t_3 e_1)$ are small, linearly independent, and in the lattice, so we hope the two short linearly independent vectors found by lattice reduction span the same sublattice as v_1 and v_2, and we can recover v_1 and v_2 by finding small points in the sublattice where coordinates 3 and 2 respectively are 0. This can be accomplished with a lattice attack. This larger lattice gives slightly improved bounds, but we omit a more detailed analysis.

7 Recovering Plaintexts

By combining the capabilities of Sections 3 through 6, the adversary can learn $\delta_{i,j} = pt_i - pt_j$ for any pair (i, j) of plaintext blocks (up to sign). It suffices that recovering any single

plaintext pt_i therefore reveals any other plaintext $pt_j = pt_i - \delta_{i,j}$. To accomplish this, we make use of the fact that $\mathrm{l}(q)$ is 2 bytes of known plaintext and a property of the RSA equations.

When the public modulus e is small, it is easy to compute the most significant bits of the private modulus d. The least significant bits of d are not easy to compute, so this does not impact the security of RSA. To see why this is the case, observe that the RSA equation implies

$$\begin{aligned} d &\equiv e^{-1} \pmod{(p-1)(q-1)} \\ \Rightarrow ed - 1 &\equiv 0 \pmod{(p-1)(q-1)} \\ \Rightarrow ed - 1 &= k(p-1)(q-1) \\ \Rightarrow k &= e\frac{d}{(p-1)(q-1)} - \frac{1}{(p-1)(q-1)} \\ \Rightarrow k &\leq e. \end{aligned}$$

Thus if e is small, all possible values of k can be brute forced. A typical choice of e is 65537, which leads to an easy brute force attack. MEGA's web client uses $e = 257$, and the SDK uses $e = 17$, so brute forcing k is even easier in this scenario.

If k is known, then

$$\begin{aligned} d &= (k(p-1)(q-1)+1)/e \\ &= (k(pq-(p+q)+1)+1)/e \\ &= \frac{kN+k+1}{e} - \frac{p+q}{e}. \end{aligned}$$

The second term is unknown, but it is about as small as p and q, which are about half the size of d. The first term is known and with high probability reveals the most significant bits of d.

To use this in the attack, we first recover $\delta_{18,1} = pt_{18} - pt_1$. pt_{18} contains 16 significant bytes of d and pt_1 contains the length encoding $\mathrm{l}(q)$. We guess all possible values of k from 1 to e, and for each guess, we determine what the significant bytes of d would be if that guess of k were correct. This gives a candidate value for pt_{18}, which we can use to compute a candidate pt_1. If the candidate pt_1 has valid length padding, the candidate pt_{18} may be correct. The odds of a false positive are acceptably small, around $e/2^{16}$, so for small e this is likely to reveal the true value of pt_{18}. Once pt_{18} is known, this reveals pt_j for every known $\delta_{18,j}$.

8 Recovering the Factorization

Section 7 demonstrates how to recover arbitrary plaintext blocks in the encoded RSA private key. This could be used to recover every plaintext block in the encoded key, but as in the attack of Backendal, Haller, and Paterson there is a more efficient solution to learning the

factorization. We can recover every plaintext block corresponding to the most significant bytes of prime factor q, then use Coppersmith's method [2] to recover the full factorization.

For the 2048-bit modulus N with 1024-bit prime factors p and q, this requires at least 512 of the most significant bits. However, there is a trade-off between how many of the most significant bits are known, how complex the implementation is, and how long it takes the implementation to run. The proof-of-concept code for the original attack requires 683 bits. We improve the implementation to be able to recover the factorization with only 624 most significant bits. This corresponds to the most significant bits of q encoded in the first 5 plaintext blocks $pt_1, pt_2, \ldots, pt_5$ of the encoded private key. With the improved implementation, recovering these 5 plaintext values suffices to recover the full factorization.

9 Complexity

In this section, we analyze the overall complexity of both the *fast* attack requiring an expected 17 login attempts and the *small* attack requiring an expected 6.1 login attempts. Because both of our attacks share many steps, we begin by describing the overlap.

Both approaches assume that the 43 bytes returned by the client are at a fixed location in the output of the RSA decryption function, but this is optimistic. As described in Section 2.2, the client returns bytes 2-44 when byte 2 is nonzero, and bytes 3-45 otherwise. This can be modeled as the attacker querying an oracle which has some small probability of returning an incorrect answer. For both of our approaches, we assume that all s responses from the oracle are correct. Empirically, the analysis steps succeed when this is true and fails otherwise. If the analysis fails, the RSA ciphertext is re-randomized and the entire attack is repeated, collecting s fresh oracle responses. Under the simplifying assumption that the probability the oracle returns a correct response for a particular input is independently distributed and equal to 255/256 (byte 2 is nonzero), the probability that all s responses are correct is $(255/256)^s$. Therefore the expected number of oracle queries before the full attack is successful is $s(256/255)^s$.

Both approaches also overlap in the final stages of the attack, so much of the complexity analysis is repeated. For the Coppersmith attack in Section 8 to succeed, we assume the attack has successfully recovered 5 plaintext blocks $pt_1, \ldots, pt_5$. To acquire these 5 plaintexts, Section 7 processes differences between these plaintexts and a plaintext pt_{18} involving MSBs of RSA private exponent d. That is, this part of the attack requires knowledge of $\delta_{18,1}, \ldots, \delta_{18,5}$. These 5 values are obtained using the technique of Section 6 from 5 high-quality approximations.

The way the two approaches differ is how these 5 high-quality approximations are obtained.

9.1 Fast Attack

In the fast attack, the 5 high-quality approximations are obtained by using Section 5 to refine 15 lower-quality approximations. For each high-quality approximation involving $\delta_{18,j}$,

we assume we have lower-quality approximations of $\delta_{18,j}x$, $2^{128}\delta_{18,j}x$, and $2^{256}\delta_{18,j}x$ for a fixed and unknown x.

These lower-quality approximations are obtained using the EHNP technique in Section 4.2. The advantage of this approach is that there is minimal guesswork involved, and it would still work if the 43 contiguous bytes were present at a different fixed offset. The disadvantage is that the EHNP transformation causes error bounds to grow, so more samples are needed to compensate. As input to the EHNP transformation, we require 15 algebraic relationships involving $2^{128t}\delta_{18,j}$ for $t \in \{0, 1, 2\}$ and $j \in \{1, 2, \ldots, 5\}$.

As described in Section 3, each algebraic relationship involves taking the difference between two client responses involving different manipulations of the wrapped RSA private key. This naively means that the attack could be performed with 30 client interactions, but because each $\delta_{18,j}$ involves the same plaintext block pt_{18}, one single client response can be reused in all 15 client response pairs. In particular, the shared ciphertext leads to the client decoding the u value as

$$\text{Decode}[\text{D}_{\text{AES}}(ct_{33}) \parallel \text{D}_{\text{AES}}(ct_{18}) \parallel \cdots \parallel \text{D}_{\text{AES}}(ct_{18}) \parallel \text{D}_{\text{AES}}(ct_{18}) \parallel \text{D}_{\text{AES}}(ct_{41})].$$

This results in a total of $s = 16$ error-free oracle responses sufficing to recover the RSA private key, or $16(256/255)^{16} \approx 17.03$ login attempts on average. None of the steps in this approach are particularly expensive, so the overall private key recovery is fast.

9.2 Small Attack

In the small attack, the 5 high-quality approximations are obtained by using the brute force technique described in Section 4.1. The input to the brute force technique are 5 algebraic relationships from Section 3, and brute force attempts to recover the unknown term $e'_1 - e''_1$, which can take on one of 511 values. Instead of trying all of the approximately 2^{45} possibilities, we improve the complexity by focusing on 3 algebraic relationships at a time. This gives a more tractable brute force cost of around 2^{27}.

For each possible combination of prefixes for the 3 algebraic relationships, we can attempt to use lattice basis B' in Section 6 to recover the unknown multipliers. If this attempt succeeds and yields valid multipliers, the guessed prefixes may be correct. If the attempt fails, the guessed prefixes are probably incorrect. In practice, this approach works to reliably return the correct prefixes for the 3 samples.

Next, we take 2 samples with recovered prefixes and 1 sample with an unknown prefix and repeat the brute force process to recover the unknown prefix. This is significantly faster than brute forcing prefixes for 3 samples simultaneously, and the technique is repeated until all unknown prefixes have been recovered. This results in 5 high-quality approximations from 5 algebraic relationships.

Using the same argument as in the fast attack, the 5 algebraic relationships can be obtained using 6 correct oracle responses, which happens with probability $(255/256)^6 \approx 98\%$. The expected number of oracle responses needed for a successful attack would be $6(256/255)^6 \approx 6.14$. The most expensive step is brute forcing the triple of unknown prefixes, but this step is easily parallelized.

Approach	Sample Size	Exp. Logins	Avg. Logins	Avg. Time (s)
Original [1]	10000	683	683 ± 0	9.46 ± 0.02
Fast Attack	10000	17.03	17.06 ± 0.08	5.59 ± 0.66
Small Attack	100	6.14	6.18 ± 0.20	16214 ± 522

Table 1. Average number of logins and average wall time required for each attack. The reported ranges represent a 95% confidence interval for the measured value.

10 Experiments

We benchmarked both of our new attacks [3] against the abstract proof-of-concept code of the attack in [1]. We ran all our attacks on an 88-core Intel Xeon E5-2699A processor running at 2.4 GHz. The original attack and our fast attack are single threaded, and our small attack implementation is multithreaded. We report a 95% confidence interval for each measurement in Table 1.

As expected, there is good agreement between the measurements and the expected complexity calculated in Section 9. The measured time includes the time to simulate the client-server interactions, explaining why the original attack, which includes more login attempts but fewer analysis steps, takes longer on average to perform. The small attack takes an average of 4 hours 30 minutes of wall-clock time to complete the analysis parallelized across 88 cores. Although this computational effort is not small, it is eminently tractable. We therefore conclude that the risk of these vulnerabilities was not limited to users who attempted to log in over 500 times, and instead show that users who attempted to log in at least 6 times may potentially be at risk. This illustrates the importance of updating clients to the latest patched version.

11 Acknowledgments

We thank Miro Haller for helpful discussions and providing further context.

12 Bibliography

References

1. Matilda Backendal, Miro Haller, and Kenneth G Paterson. MEGA: Malleable encryption goes awry. Preprint at `https://mega-awry.io/pdf/mega-malleable-encryption-goes-awry.pdf`, June 2022.
2. Don Coppersmith. Finding a small root of a bivariate integer equation; factoring with high bits known. In Ueli M. Maurer, editor, *Advances in Cryptology – EUROCRYPT'96*, volume 1070 of *Lecture Notes in Computer Science*, pages 178–189, Saragossa, Spain, May 12–16, 1996. Springer, Heidelberg, Germany.
3. Harvey L. Garner. The residue number system. In *Papers Presented at the the March 3-5, 1959, Western Joint Computer Conference*, IRE-AIEE-ACM '59 (Western), page 146–153, New York, NY, USA, 1959. Association for Computing Machinery.
4. Martin Hlavác and Tomás Rosa. Extended hidden number problem and its cryptanalytic applications. In Eli Biham and Amr M. Youssef, editors, *SAC 2006: 13th Annual International Workshop on Selected Areas in Cryptography*, volume 4356 of *Lecture Notes in Computer Science*, pages 114–133, Montreal, Canada, August 17–18, 2007. Springer, Heidelberg, Germany.

[3] Our updated implementation is available at `https://github.com/keeganryan/attacks-poc`

5. Nick A. Howgrave-Graham, Phong Q. Nguyen, and Igor E. Shparlinski. Hidden number problem with hidden multipliers, timed-release crypto, and noisy exponentiation. *Mathematics of Computation*, 72(243):1473–1485, 2003.
6. Mathias Ortmann. MEGA security update, Jun 2022. `https://blog.mega.io/mega-security-update/`.