NTNU | Norwegian University of Science and Technology

PROTOCOL COMPOSITION 2

TTM4205 – Lecture 16

Tjerand Silde

31.10.2023

Contents

Contents

Reminder: Special Topic Project

The deadline for submitting group and topic is **Nov 1st**.

The Remaining Schedule

44	31/10	Lecture	Tjerand	Protocol Composition 2
44	31/10	Lab/Ex	Jonathan	Composition Exercises 1
44	2/11	Lab/Ex	Jonathan	Composition Exercises 2
45	7/11	Lecture	Tjerand	Protocol Composition 3
45	7/11	Lab/Ex	Jonathan	Assignments
45	9/11	Lecture	Tjerand	Course Summary
46	14/11	Lecture	Tjerand	Guest Lecture: Håkon Jacobsen
46	14/11	Lab/Ex	Tjerand	Assignments
46	16/11	Lecture	Tjerand	Guest Lecture: Oskar Goldhahn
47	21/11	Lab/Ex	Jonathan	Assigments
47	21/11	Lecture	Tjerand	Guest Lecture: Vadim Lyubashevsky
47	23/11	Lecture	Tjerand	***Project Presentations***

Reference Group Meeting

Summary.

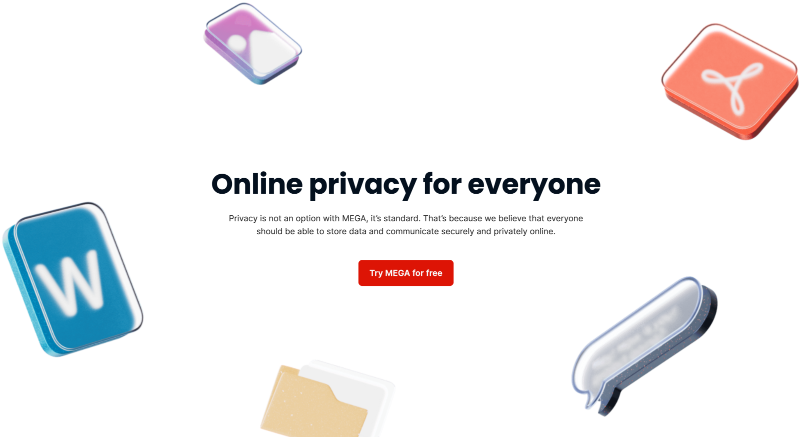
Invited Talk Today

Invited talk by Matthias Wichtlhuber (DE-CIX) from 15:00-16:00 in Realfagbygget R1 today on:

"DDoS Defense at Scale: Automated Training Data Generation for ML-Based Protection at DE-CIX".

Contents

MEGA E2EE Cloud Storage



MEGA E2EE Cloud Storage

- The user encrypt all files locally
- It upload ciphertexts to the cloud
- File-keys are encrypted under master-key
- Master-key is encrypted under password
- The user can log in from anywhere
- The user must sign a challenge on log-in

This is initially a secure infrastructure, but we will see that 1) the choice of ciphers, 2) how they are composed, 3) the lack of integrity checks, 4) custom padding, 5) key-reuse, and 6) server-chosen plaintexts breaks the security in many ways.

Contents

MEGA: Malleable Encryption Goes Awry

Matilda Backendal, Miro Haller and Kenneth G. Paterson
Department of Computer Science, ETH Zurich, Zurich, Switzerland
Email: {mbackendal, kenny.paterson}@inf.ethz.ch, miro.haller@alumni.ethz.ch

Figure: https://eprint.iacr.org/2022/959.pdf

MEGA: MALLEABLE ENCRYPTION GOES AWRY

MEGA is a leading cloud storage platform with more than 250 million users and 1000 Petabytes of stored data, which aims to achieve user-controlled end-to-end encryption. We show that MEGA's system does not protect its users against a malicious server and present five distinct attacks, which together allow for a full compromise of the confidentiality of user files. Additionally, the integrity of user data is damaged to the extent that an attacker can insert malicious files of their choice which pass all authenticity checks of the client. We built proof-of-concept versions of all the attacks, showcasing their practicality and exploitability.

Figure: `https://mega-awry.io`

Figure: https://twitter.com/kennyog/status/1539352663770509314

Attacks

Attacks

RSA Key Recovery Attack

MEGA can recover a user's RSA private key by maliciously tampering with 512 login attempts.

Plaintext Recovery

MEGA can decrypt other key material, such as node keys, and use them to decrypt all user communication and files.

Framing Attack

MEGA can insert arbitrary files into the user's file storage which are indistinguishable from genuinely uploaded ones.

Integrity Attack

The impact of this attack is the same as that of the framing attack, trading off less stealthiness for easier pre-requisites.

GaP-Bleichenbacher Attack

MEGA can decrypt RSA ciphertexts using an expensive padding oracle attack.

RSA Key Recovery Attack

RSA Key Recovery Attack

- k_M is an AES key derived from a password

RSA Key Recovery Attack

- k_M is an AES key derived from a password
- $\mathsf{pk}_{\mathsf{share}}$ is a RSA public key $(N = p \cdot q, e)$

RSA Key Recovery Attack

- k_M is an AES key derived from a password
- $\mathsf{pk}_{\mathsf{share}}$ is a RSA public key $(N = p \cdot q, e)$
- $\mathsf{sk}_{\mathsf{share}}^{\mathsf{encoded}}$ is the corresponding secret key

RSA Key Recovery Attack

- k_M is an AES key derived from a password
- $\mathsf{pk}_{\mathsf{share}}$ is a RSA public key $(N = p \cdot q, e)$
- $\mathsf{sk}_{\mathsf{share}}^{\mathsf{encoded}}$ is the corresponding secret key
- $[\mathsf{sk}_{\mathsf{share}}^{\mathsf{encoded}}]_{k_M}$ is AES encrypted under k_M

RSA Key Recovery Attack

- k_M is an AES key derived from a password
- $\mathsf{pk}_{\mathsf{share}}$ is a RSA public key $(N = p \cdot q, e)$
- $\mathsf{sk}_{\mathsf{share}}^{\mathsf{encoded}}$ is the corresponding secret key
- $[\mathsf{sk}_{\mathsf{share}}^{\mathsf{encoded}}]_{k_M}$ is AES encrypted under k_M
- the client is given $[\mathsf{sk}_{\mathsf{share}}^{\mathsf{encoded}}]_{k_M}$ at every log-in

RSA Key Recovery Attack

- k_M is an AES key derived from a password
- $\mathsf{pk}_{\mathsf{share}}$ is a RSA public key $(N = p \cdot q, e)$
- $\mathsf{sk}^{\mathsf{encoded}}_{\mathsf{share}}$ is the corresponding secret key
- $[\mathsf{sk}^{\mathsf{encoded}}_{\mathsf{share}}]_{k_M}$ is AES encrypted under k_M
- the client is given $[\mathsf{sk}^{\mathsf{encoded}}_{\mathsf{share}}]_{k_M}$ at every log-in
- the client is also given $[m]_{\mathsf{pk}_{\mathsf{share}}}$ at log-in

RSA Key Recovery Attack

- k_M is an AES key derived from a password
- pk_share is a RSA public key $(N = p \cdot q, e)$
- $\text{sk}_\text{share}^\text{encoded}$ is the corresponding secret key
- $[\text{sk}_\text{share}^\text{encoded}]_{k_M}$ is AES encrypted under k_M
- the client is given $[\text{sk}_\text{share}^\text{encoded}]_{k_M}$ at every log-in
- the client is also given $[m]_{\text{pk}_\text{share}}$ at log-in
- m is a randomly sampled 43 B session ID

RSA Key Recovery Attack

The master secret key is encoded in the following way:

$$\mathsf{sk}_{\mathsf{share}}^{\mathsf{encoded}} \leftarrow l(q)||q||l(p)||p||l(d)||d||l(u)||u||P$$

where $l(\cdot)$ is a length function, q and p are 1024-bit primes, d is the secret RSA exponent, $u = q^{-1} \bmod p$ and P is padding.

RSA Key Recovery Attack

The following happens when the client log in:

RSA Key Recovery Attack

The following happens when the client log in:

- The client derives k_M locally from password

RSA Key Recovery Attack

The following happens when the client log in:

- The client derives k_M locally from password
- The server sends stored $[\mathsf{sk}^{\mathsf{encoded}}_{\mathsf{share}}]_{k_M}$ to client

RSA Key Recovery Attack

The following happens when the client log in:

- The client derives k_M locally from password
- The server sends stored $[\mathsf{sk}_{\mathsf{share}}^{\mathsf{encoded}}]_{k_M}$ to client
- The server samples m and sends $[m]_{\mathsf{pk}_{\mathsf{share}}}$ to client

RSA Key Recovery Attack

The following happens when the client log in:

- The client derives k_M locally from password
- The server sends stored $[\mathsf{sk}_{\mathsf{share}}^{\mathsf{encoded}}]_{k_M}$ to client
- The server samples m and sends $[m]_{\mathsf{pk}_{\mathsf{share}}}$ to client
- The client decrypts $[\mathsf{sk}_{\mathsf{share}}^{\mathsf{encoded}}]_{k_M}$ then decrypts $[m]_{\mathsf{pk}_{\mathsf{share}}}$

RSA Key Recovery Attack

The following happens when the client log in:

- The client derives k_M locally from password
- The server sends stored $[\mathsf{sk}_{\mathsf{share}}^{\mathsf{encoded}}]_{k_M}$ to client
- The server samples m and sends $[m]_{\mathsf{pk}_{\mathsf{share}}}$ to client
- The client decrypts $[\mathsf{sk}_{\mathsf{share}}^{\mathsf{encoded}}]_{k_M}$ then decrypts $[m]_{\mathsf{pk}_{\mathsf{share}}}$
- The client sends m to the server which accepts/rejects

RSA Key Recovery Attack

The following happens when the client log in:

- The client derives k_M locally from password
- The server sends stored $[\mathsf{sk}_{\mathsf{share}}^{\mathsf{encoded}}]_{k_M}$ to client
- The server samples m and sends $[m]_{\mathsf{pk}_{\mathsf{share}}}$ to client
- The client decrypts $[\mathsf{sk}_{\mathsf{share}}^{\mathsf{encoded}}]_{k_M}$ then decrypts $[m]_{\mathsf{pk}_{\mathsf{share}}}$
- The client sends m to the server which accepts/rejects
- The server sends all encrypted files to client if accept

$\mathsf{DecSid}([sk_{share}^{encoded}]_{k_M}, [m]_{pk_{share}})$:

Given: encrypted RSA private key $[sk_{share}^{encoded}]_{k_M}$, encrypted message $[m]_{pk_{share}}$
Returns: decrypted and unpadded SID sid'

$sk_{share}^{encoded} \leftarrow \mathsf{AES\text{-}ECB.Dec}(k_M, [sk_{share}^{encoded}]_{k_M})$
$N, e, d, p, q, d_p, d_q, u \leftarrow \mathsf{DecodeRsaKey}(sk_{share}^{encoded})$
$m'_p \leftarrow ([m]_{pk_{share}})^{d_p} \bmod p$
$m'_q \leftarrow ([m]_{pk_{share}})^{d_q} \bmod q$
$t \leftarrow m'_p - m'_q \bmod p$
$h \leftarrow t \cdot u \bmod p$
$m' \leftarrow h \cdot q + m'_q$
$sid' \leftarrow m'[3{:}45]$ // Unpad 43 B SID.
return sid'

Fig. 5. SID decryption during MEGA's client authentication using RSA.

RSA Key Recovery Attack

We can break the system in the following way:

RSA Key Recovery Attack

We can break the system in the following way:

- the secret key $\mathsf{sk}_{\mathsf{share}}^{\mathsf{encoded}}$ is encrypted with AES-ECB

RSA Key Recovery Attack

We can break the system in the following way:

- the secret key $\mathsf{sk}_{\mathsf{share}}^{\mathsf{encoded}}$ is encrypted with AES-ECB
- there is no integrity check for $[\mathsf{sk}_{\mathsf{share}}^{\mathsf{encoded}}]_{k_M}$

RSA Key Recovery Attack

We can break the system in the following way:

- the secret key $\mathsf{sk}_{\mathsf{share}}^{\mathsf{encoded}}$ is encrypted with AES-ECB
- there is no integrity check for $[\mathsf{sk}_{\mathsf{share}}^{\mathsf{encoded}}]_{k_M}$
- we can edit $[\mathsf{sk}_{\mathsf{share}}^{\mathsf{encoded}}]_{k_M}$ so that only u changes

RSA Key Recovery Attack

We can break the system in the following way:

- the secret key $\mathsf{sk}_{\mathsf{share}}^{\mathsf{encoded}}$ is encrypted with AES-ECB
- there is no integrity check for $[\mathsf{sk}_{\mathsf{share}}^{\mathsf{encoded}}]_{k_M}$
- we can edit $[\mathsf{sk}_{\mathsf{share}}^{\mathsf{encoded}}]_{k_M}$ so that only u changes
- recover parts of q from chosen m with faulty u

RSA Key Recovery Attack

We can break the system in the following way:

- the secret key $\mathsf{sk}_{\mathsf{share}}^{\mathsf{encoded}}$ is encrypted with AES-ECB
- there is no integrity check for $[\mathsf{sk}_{\mathsf{share}}^{\mathsf{encoded}}]_{k_M}$
- we can edit $[\mathsf{sk}_{\mathsf{share}}^{\mathsf{encoded}}]_{k_M}$ so that only u changes
- recover parts of q from chosen m with faulty u
- decrypt all files that the client stored under $[\mathsf{sk}_{\mathsf{pk}_{\mathsf{AES}}}]_{\mathsf{pk}_{\mathsf{share}}}$

RSA Key Recovery Attack

We can recover q from faulty u as follows:

RSA Key Recovery Attack

We can recover q from faulty u as follows:

- If $m < q$ then we get $m'_p = m = m'_q$

RSA Key Recovery Attack

We can recover q from faulty u as follows:

- If $m < q$ then we get $m'_p = m = m'_q$
- Then $t = 0$ and $h = 0$ and $m' = m < 256^{128}$

RSA Key Recovery Attack

We can recover q from faulty u as follows:

- If $m < q$ then we get $m'_p = m = m'_q$
- Then $t = 0$ and $h = 0$ and $m' = m < 256^{128}$
- m' is padded with zeros to 256 bytes

RSA Key Recovery Attack

We can recover q from faulty u as follows:

- If $m < q$ then we get $m'_p = m = m'_q$
- Then $t = 0$ and $h = 0$ and $m' = m < 256^{128}$
- m' is padded with zeros to 256 bytes
- Remove the 211 rightmost bytes

RSA Key Recovery Attack

We can recover q from faulty u as follows:

- If $m < q$ then we get $m'_p = m = m'_q$
- Then $t = 0$ and $h = 0$ and $m' = m < 256^{128}$
- m' is padded with zeros to 256 bytes
- Remove the 211 rightmost bytes
- Then the returned $\mathsf{sid} = \mathsf{m}[3:45] = 0$

RSA Key Recovery Attack

We can recover q from faulty u as follows:

RSA Key Recovery Attack

We can recover q from faulty u as follows:

- If $m \geq q$ then we get $m'_p \neq m \neq m'_q$

RSA Key Recovery Attack

We can recover q from faulty u as follows:

- If $m \geq q$ then we get $m'_p \neq m \neq m'_q$
- Then $t \neq 0$ and $h \neq 0$ since $u \neq q^{-1} \bmod p$

RSA Key Recovery Attack

We can recover q from faulty u as follows:

- If $m \geq q$ then we get $m'_p \neq m \neq m'_q$
- Then $t \neq 0$ and $h \neq 0$ since $u \neq q^{-1} \bmod p$
- Then $m' \neq m > 256^{128}$ with high prob.

RSA Key Recovery Attack

We can recover q from faulty u as follows:

- If $m \geq q$ then we get $m'_p \neq m \neq m'_q$
- Then $t \neq 0$ and $h \neq 0$ since $u \neq q^{-1} \bmod p$
- Then $m' \neq m > 256^{128}$ with high prob.
- Remove the 211 rightmost bytes

RSA Key Recovery Attack

We can recover q from faulty u as follows:

- If $m \geq q$ then we get $m'_p \neq m \neq m'_q$
- Then $t \neq 0$ and $h \neq 0$ since $u \neq q^{-1} \bmod p$
- Then $m' \neq m > 256^{128}$ with high prob.
- Remove the 211 rightmost bytes
- Then the returned $\mathsf{sid} = \mathsf{m}[3:45] \neq 0$

RSA Key Recovery Attack

This means that we learn 1 bit of information each time, and can use a binary search between $[2^{1023}, 2^{1024})$ to find q in at most 1023 queries i.e. each time the client tries to log in.

Using an improved lattice-attack similar to the attack on ECDSA allowed for a reduction to 512 queries total.

Other Attacks

The re-use of keys also allowed for decryption oracles, the custom RSA padding P allowed for Bleichenbacher attacks, lack of integrity checks allowed for uploading malicious material, and more.

They added HMAC checks, updated padding and updated the key-hierarchy after this work, but claimed that 512 log-in attempts was too much for this to be a realistic attack...

Contents

The Hidden Number Problem with Small Unknown Multipliers: Cryptanalyzing MEGA in Six Queries and Other Applications

Keegan Ryan and Nadia Heninger

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

Figure: https://eprint.iacr.org/2022/914.pdf

We actually learn more than 1 bit per query. The sid is of size 43 bytes, and this leaks much more information. With a lot of pre-processing, it was shown that we can recover q in only 6 queries (!). The lattice-attack achieving this is out of scope.

Contents

Caveat Implementor! Key Recovery Attacks on MEGA

Martin R. Albrecht[1], Miro Haller[2], Lenka Mareková[3], and Kenneth G. Paterson[2]

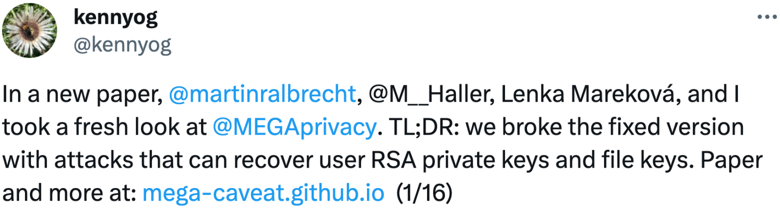
[1] King's College London
martin.albrecht@kcl.ac.uk
[2] Applied Cryptography Group, ETH Zurich
kenny.paterson@inf.ethz.ch, miro.haller@ethz.ch
[3] Information Security Group, Royal Holloway, University of London
lenka.marekova.2018@rhul.ac.uk

Figure: https://eprint.iacr.org/2023/329.pdf

Caveat Implementor!
Key Recovery Attacks on MEGA

MEGA is a large-scale cloud storage and communication platform that aims to provide end-to-end encryption for stored data. Recent work by Backendal, Haller and Paterson invalidated these security claims by showing practical attacks against MEGA that could be mounted by the MEGA service provider. In response, the MEGA developers added lightweight sanity checks on the user RSA private keys used in MEGA, sufficient to prevent the previous attacks. We analysed these new sanity checks and show how they themselves could be exploited to mount novel attacks on MEGA that recover a target user's RSA private key with only slightly higher attack complexity than the original attacks.

Figure: `https://mega-caveat.github.io`


kennyog
@kennyog

In a new paper, @martinralbrecht, @M__Haller, Lenka Mareková, and I took a fresh look at @MEGAprivacy. TL;DR: we broke the fixed version with attacks that can recover user RSA private keys and file keys. Paper and more at: mega-caveat.github.io (1/16)

Figure:
`https://twitter.com/kennyog/status/1632718211476078592`

New Attacks

They came up with the following new attacks:

New Attacks

They came up with the following new attacks:

- Reducing the original attack from 512 to 2 queries

New Attacks

They came up with the following new attacks:

- Reducing the original attack from 512 to 2 queries
- Exploiting re-encryption with adversarial keys

New Attacks

They came up with the following new attacks:

- Reducing the original attack from 512 to 2 queries
- Exploiting re-encryption with adversarial keys
- Error messages that reveal more information

New Attacks

They came up with the following new attacks:

- Reducing the original attack from 512 to 2 queries
- Exploiting re-encryption with adversarial keys
- Error messages that reveal more information
- Still using AES-ECB because it is "cheaper"

New Attacks

They came up with the following new attacks:

- Reducing the original attack from 512 to 2 queries
- Exploiting re-encryption with adversarial keys
- Error messages that reveal more information
- Still using AES-ECB because it is "cheaper"
- Key-overwriting attacks from lacking integrity

Questions?