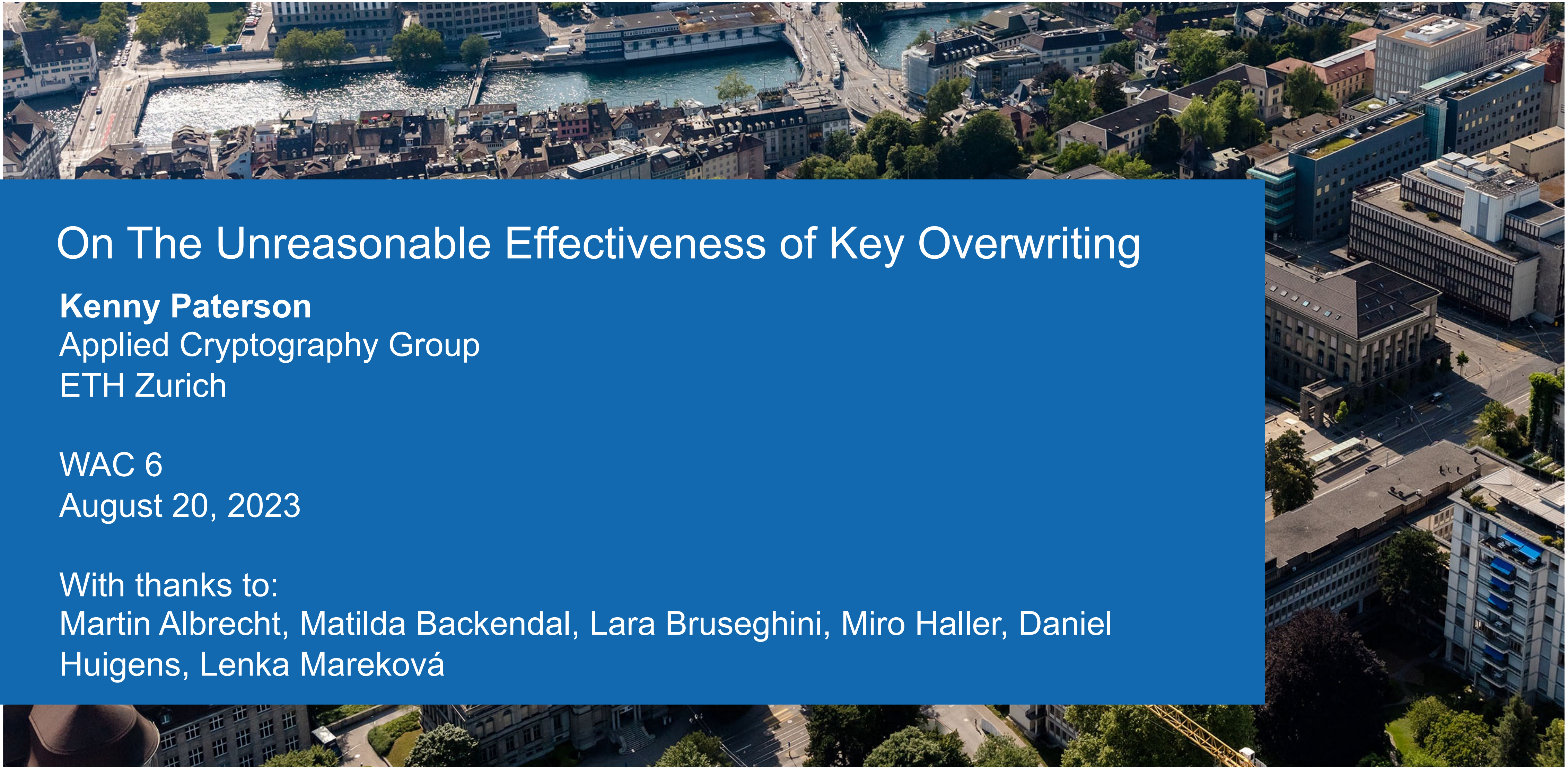


On The Unreasonable Effectiveness of Key Overwriting

Kenny Paterson
Applied Cryptography Group
ETH Zurich

WAC 6
August 20, 2023

With thanks to:
Martin Albrecht, Matilda Backendal, Lara Bruseghini, Miro Haller, Daniel Huigens, Lenka Mareková

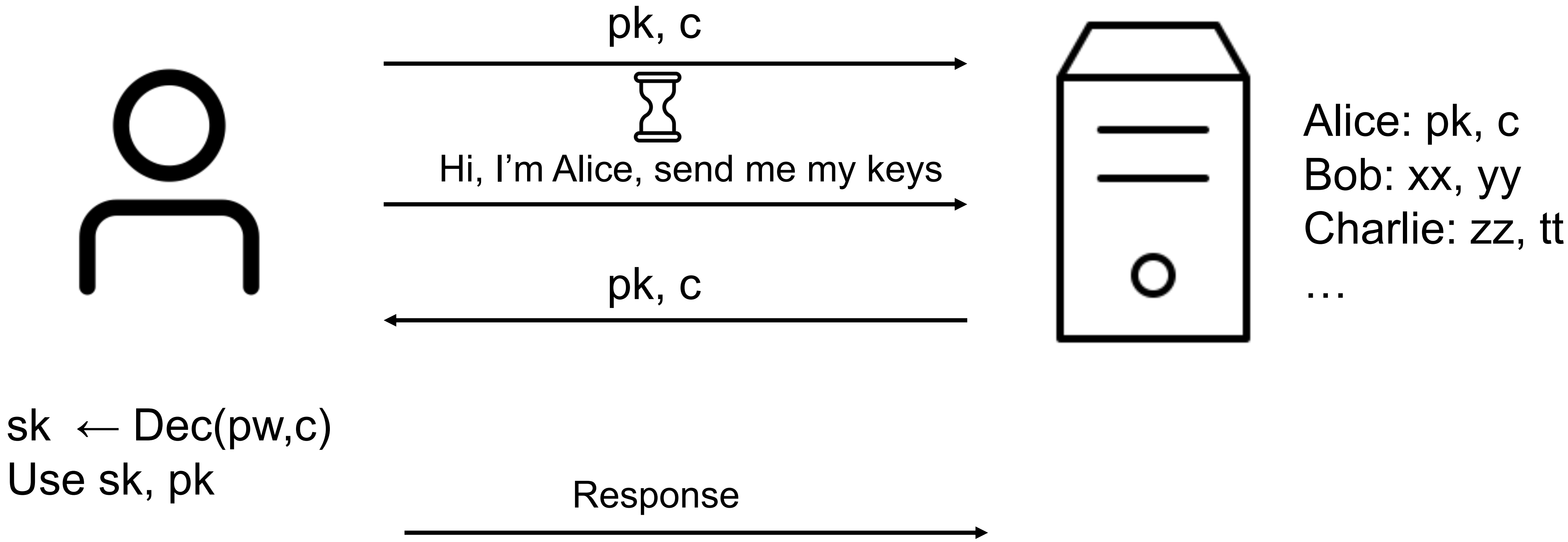
Overview

- Introducing Key Overwriting Attacks
- Easier: OpenPGP
- Harder: MEGA
- Wrap-up

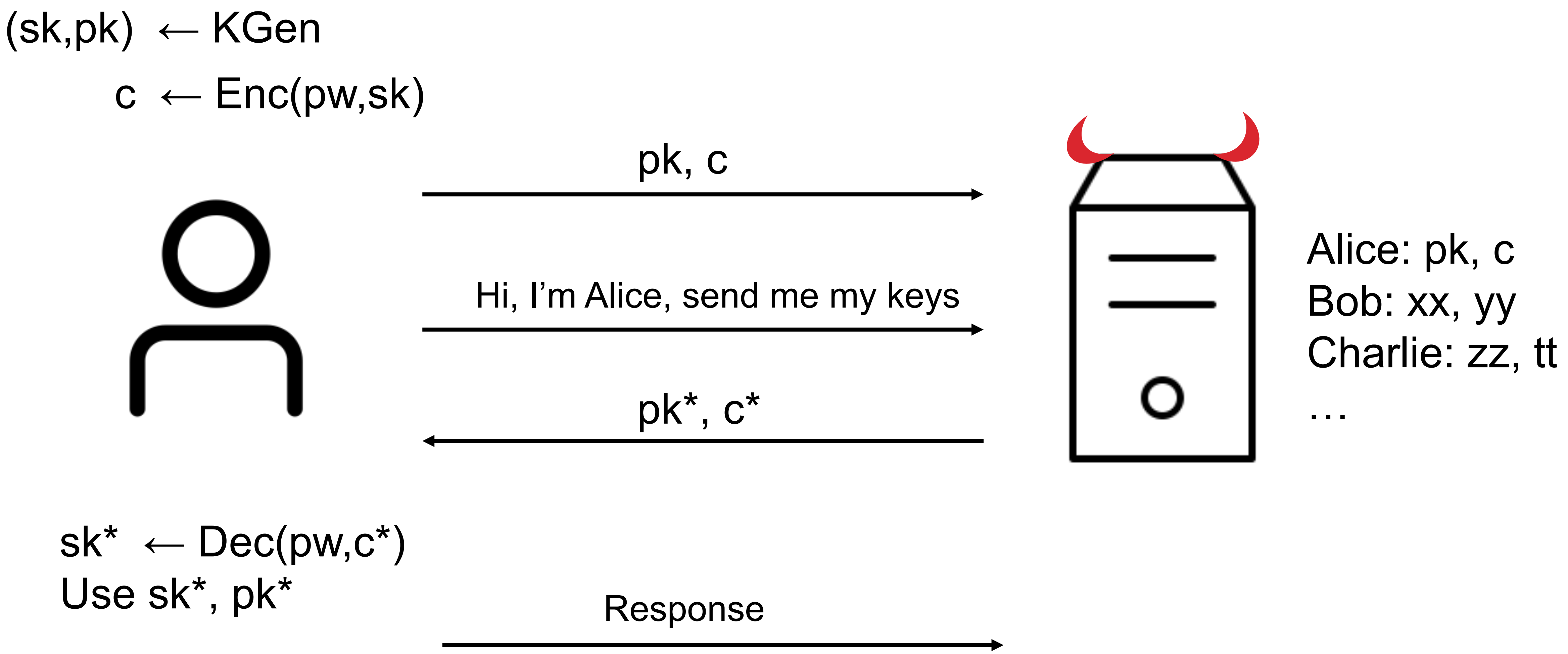
Introducing Key Overwriting (KO) Attacks

$(sk, pk) \leftarrow KGen$

$c \leftarrow Enc(pw, sk)$



Introducing Key Overwriting (KO) Attacks



Introducing Key Overwriting (KO) Attacks

- In a key overwriting attack, the adversary is able to **overwrite** part or all of the victim's keys AND **observe** what happens when the modified keys are subsequently used.
- Adversary targets recovery of original keys (or maybe has a weaker objective).
- Mostly relevant in public key setting, but also applies in symmetric setting.
- Overwriting may be **controlled** or **uncontrolled** (or somewhere in-between).
- Overwriting may be limited **to public key only**, or to **private key only**, or be possible for both components.
- KO attacks are related to fault attacks, related key attacks, and memory tampering attacks, cf. Bellcore attack on CRT-RSA.

Introducing Key Overwriting (KO) Attacks

- Adversary may be able to repeatedly overwrite keys with adaptively chosen values.
- Its observation capability may be limited.
- Attacks may involve user interaction, which we try to minimise.
- Keys may be **validated** by client software before use: increasing the complexity of attacks…

… or introducing new attack vectors!

Introducing Key Overwriting (KO) Attacks

APPLIED CRYPTO GROUP

Thesis of this talk:

KO attacks are a powerful weapon in the attacker's armoury that we should learn how to use!

Evidence:

- V. Klíma, T. Rosa. Attack on Private Signature Keys of the OpenPGP Format, PGP (TM) Programs and Other Applications Compatible with OpenPGP. IACR Cryptology ePrint Archive 2002/76.
- L. Bruseghini, K.G. Paterson, D. Huigens. Victory by KO: Attacking OpenPGP Using Key Overwriting, ACM CCS 2022.
- M. Backendal, M. Haller, K.G. Paterson. MEGA: Malleable Encryption Goes Awry, IEEE S&P 2023.
- M.R. Albrecht, M. Haller, L. Mareková, K.G. Paterson. *Caveat Implementor!* Key Recovery Attacks on MEGA, Eurocrypt 2023.

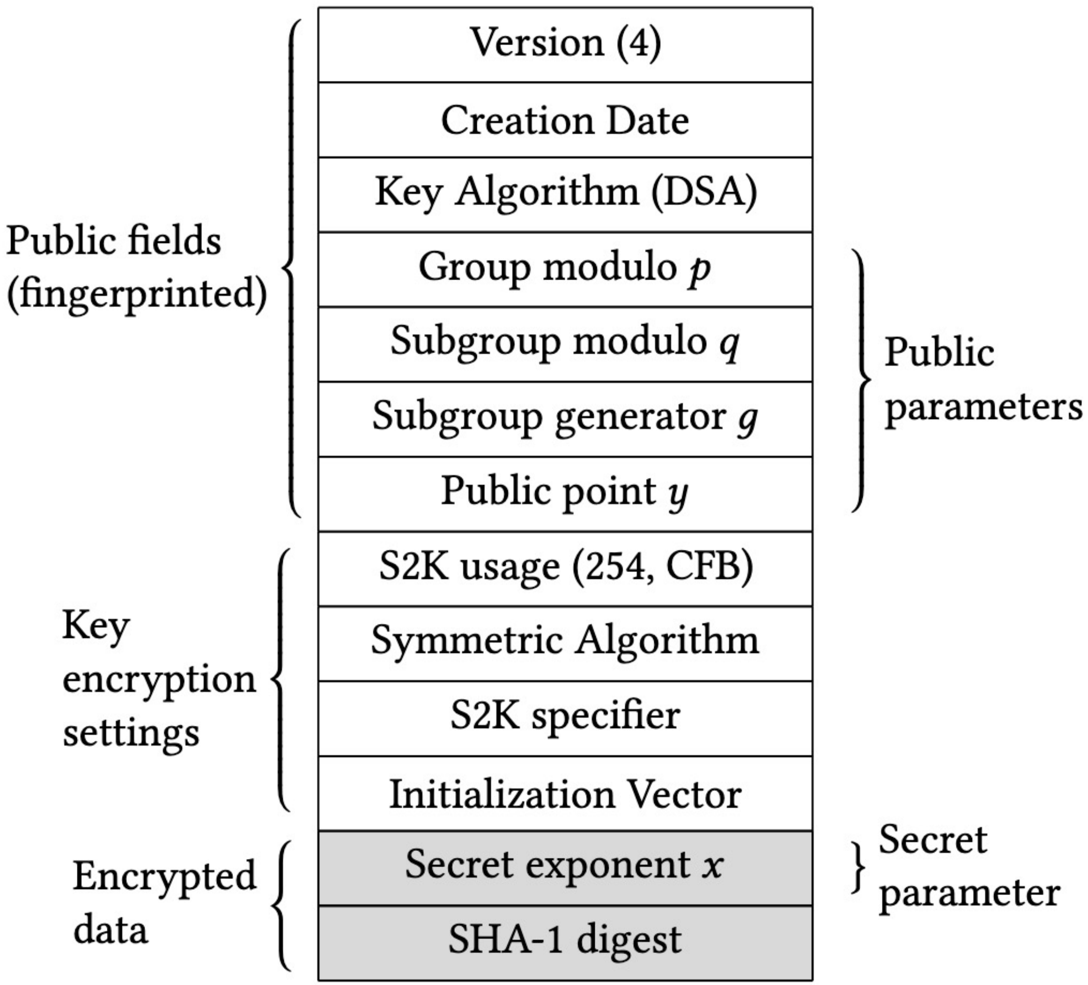
Overview

- Introducing Key Overwriting Attacks
- **Easier: OpenPGP**
- Harder: MEGA
- Wrap-up

Public Key Setting – OpenPGP

- Widely-used email encryption standard, starting with RFC 2440.
- Increasingly used in cloud-based solutions with outsourced key storage, e.g. ProtonMail, FlowCrypt.
- Now becoming used beyond email too, e.g. in Proton's secure storage solution.
- Rich community of developers and different implementations:
  - GPG, Sequoia, RNP, OpenPGP.js, gopenpgp,…
- Lot of legacy cryptography supported in OpenPGP.
  - Has led to attacks, e.g. under-specified ElGamal encryption [DPS2021].
- OpenPGP crypto refresh now being worked on in IETF.
  - https://datatracker.ietf.org/doc/html/draft-ietf-openpgp-crypto-refresh

Example OpenPGP Key Format: DSA



White fields: not cryptographically protected.

Grey fields: protected using “hash-then-encrypt” mechanism, so confidentiality and some degree of integrity.

→

We are in the setting where only the public key can be overwritten.

DSA in OpenPGP

Algorithm 1: DSA signing and verification

Data: Message m, public key (g, p, q, y), private key x, signature (r, s)

Function `DSA_Sign`$(m, (g, p, q), x)$

$k \xleftarrow{\$} [1, q-1]$

$r = (g^k \bmod p) \bmod q$

$s = k^{-1}(\text{Hash}(m) + xr) \bmod q$

return (r, s)

Function `DSA_Verify`$(m, (g, p, q, y), (r, s))$

$w = s^{-1} \bmod q$

$h = \text{Hash}(m) \bmod q$

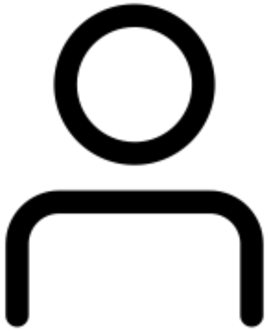
$v = (g^{hw} y^{rw} \bmod p) \bmod q$

return $v \stackrel{?}{=} r$

Klíma and Rosa Attack on OpenPGP

$(sk=x,\ pk=(g,p,q,y=g^x)) \leftarrow$ DSA.KGen

$c \leftarrow \mathrm{Enc}(pw,x)$



pk, c →

Hi, I'm Alice, send me my key →

← $pk^* = (g^*,p^*,q^*,y)$, c

Alice: pk, c

Select g^*,p^*,q^* such that:

1. p^* is prime with p^*-1 smooth
2. $q^* > p^*$ (!!)
3. g^* generates $(Z_{p^*})^*$

$x \leftarrow \mathrm{Dec}(pw,c)$

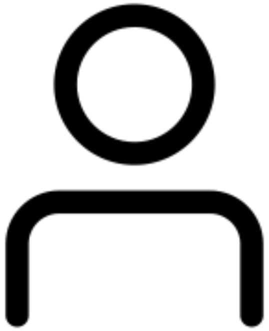
$(r,s) \leftarrow$ DSA.Sign$(x,(g^*,p^*,q^*),m)$

(r,s) →

Klíma and Rosa Attack on OpenPGP

$(sk=x,\ pk=(g,p,q,y=g^x)) \leftarrow$ DSA.KGen

$c \leftarrow Enc(pw,x)$



pk, c →

Hi, I'm Alice, send me my key →

← pk* = (g*,p*,q*,y), c

Alice: pk, c

$x \leftarrow Dec(pw,c)$

$(r,s) \leftarrow$ DSA.Sign(x,(g*,p*,q*),m)

(r,s) →

$r = ((g^*)^k \bmod p^*) \bmod q^*$
$= (g^*)^k \bmod p^*$

1. Recover k from r by solving DLP in easy DLP group $(Z_{p^*})^*$.
2. Recover x from:
   $s = k^{-1}(H(m)+xr) \bmod q^*$

Klíma and Rosa Attack – Summary

Select g^*, p^*, q^* such that:

1. p^* is prime with p^*-1 smooth
2. $q^* > p^*$ (!!)
3. g^* generates $(Z_{p^*})^*$

$r = ((g^*)^k \bmod p^*) \bmod q^*$
$\quad = (g^*)^k \bmod p^*$

1. Recover k by solving DLP in easy DLP group $(Z_{p^*})^*$.
2. Recover x from:
   $s = k^{-1}(H(m)+xr) \bmod q^*$

- Extraction of the private key x from a single faulty signature!
- Somewhat artificial because of large q^* (recall q^* is typically small, e.g. 160-256 bits).
- Attack can be prevented by careful validation of (sk,pk) (but key validation is not required or specified by OpenPGP).
- Most libraries are not vulnerable to this attack today because of restrictions on parameter sizes.

Key Overwriting Attacks Against OpenPGP

APPLIED CRYPTO GROUP

Victory by KO: Attacking OpenPGP Using Key Overwriting*

Lara Bruseghini
ETH Zurich and Proton AG
larabr@protonmail.com

Kenneth G. Paterson
Applied Cryptography Group, ETH Zurich
kenny.paterson@inf.ethz.ch

Daniel Huigens
Proton AG
d.huigens@protonmail.com

ABSTRACT

We present a set of attacks on the OpenPGP specification and implementations of it which result in full recovery of users' private keys. The attacks exploit the lack of cryptographic binding between the different fields inside an encrypted private key packet, which include the key algorithm identifier, the cleartext public parameters, and the encrypted private parameters. This allows an attacker who

downloading and sending messages, and remote parties do not get to communicate directly with cryptographic software, but where that software is only used locally to decrypt/encrypt or sign/verify some emails. However, the use cases for OpenPGP have evolved, and application scenarios have changed over the past 20 years. In particular, we now see widespread use of cloud-based storage, in-browser and server-provided encryption services, and automated

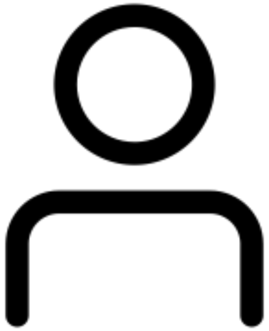
ACM CCS 2022
https://www.kopenpgp.com/

Klíma and Rosa Rebooted

(sk=x, pk=(g,p,q,y=g^x)) ← DSA.KGen

c ← Enc(pw,x)

pk, c →



Hi, I'm Alice, send me my key →

Alice: pk, c

← pk* = (g*,p*,q*,y), c

Select g*, p*, q* such that:

1. p* is prime
2. q* is small (e.g. 16 bits) and divides p*-1
3. g* generates an order q* subgroup of $(Z_{p^*})^*$

x ← Dec(pw,c)

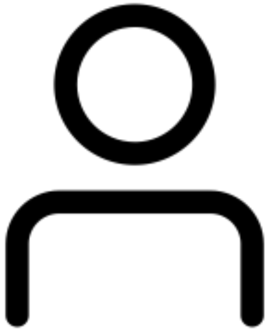
(r,s) ← DSA.Sign(x,(g*,p*,q*),m)

(r,s) →

Klíma and Rosa Rebooted

$(sk=x,\ pk=(g,p,q,y=g^x)) \leftarrow$ DSA.KGen

$c \leftarrow Enc(pw,x)$



pk, c →

Hi, I'm Alice, send me my key →

Alice: pk, c

← $pk^* = (g^*,p^*,q^*,y)$, c

$x \leftarrow Dec(pw,c)$

$(r,s) \leftarrow DSA.Sign(x,(g^*,p^*,q^*),m)$

(r,s) →

$r = ((g^*)^k \bmod p^*) \bmod q^*$
$s = k^{-1}(H(m)+xr) \bmod q^*$

w.h.p. there is exactly one value $y=(g^*)^x \bmod p^*$ for which (r,s) verifies against pk*.

- Offline, try all q* possible values to recover x mod q*
- Work with several different q*; recover x via CRT.

Cross Algorithm Attack on OpenPGP

APPLIED CRYPTO GROUP

(sk=x, pk =[x]P)) ← ECC.KGen

c ← Enc(pw,x)

pk, c →

Alice: pk, c

Hi, I'm Alice, send me my key →

← pk* = DSA.pk(g*,p*,q*,y), c

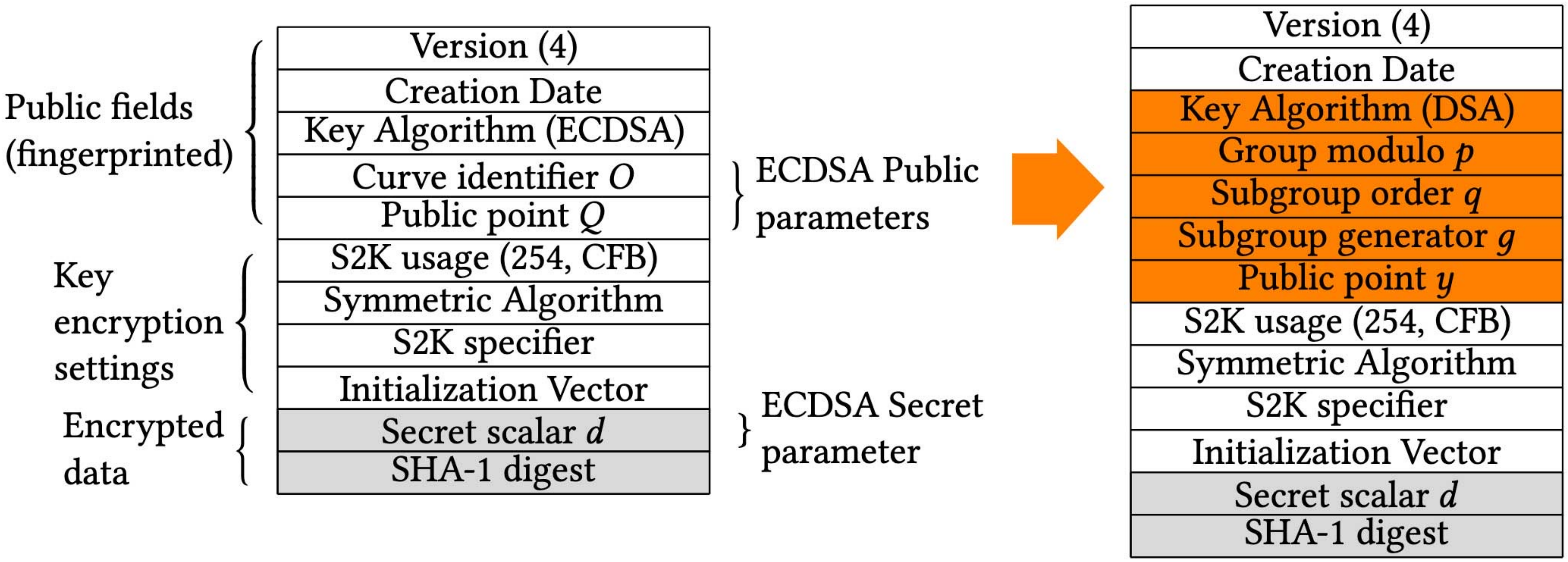
x ← Dec(pw,c)

(r,s) ← DSA.Sign(x,(g*,p*,q*),m)

(r,s) →

Run either of the previous attacks to recover x!

Cross Algorithm Attack on OpenPGP

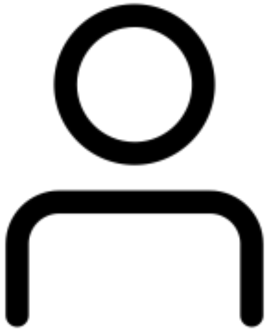


Key Overwriting with Key Validation (KOKV)

APPLIED CRYPTO GROUP

(sk, pk) ← KGen

c ← Enc(pw,x)



pk, c →

Hi, I'm Alice, send me my key →

← pk*, c*

Alice: pk, c

sk* ← Dec(pw,c*)

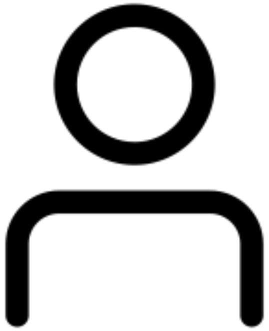
Validate sk*, pk* by some means

Success/failure →

KOKV Attack for DSA in GPG/libgcrypt

$(sk=x, pk=(g,p,q,y=g^x)) \leftarrow$ DSA.KGen

$c \leftarrow Enc(pw,x)$



pk, c →

Hi, I'm Alice, send me my key →

← $pk^* = (g^*,p^*,q^*,y^*)$, c

Alice: pk, c

$x \leftarrow Dec(pw,c)$

Check if $(g^*)^x = y^* \bmod p^*$

Success/failure →

(see next slide)

KOKV Attack for DSA in GPG/libgcrypt

- Denote unknown bits of x by x_0, x_1, x_2,… (from LSB up).

- Set $p^* = 2^t \cdot h + 1$, set g^* of order 2 mod p^*, i.e. $g^* = -1 \bmod p^*$.
- Now key validation takes place in a group of order 2 and only x mod 2 is relevant!

- Set $y^* = 1 = (g^*)^0 \bmod p^*$.
- Then $(g^*)^x = y^* \bmod p^* \Leftrightarrow x = 0 \bmod 2$.
- That is, key validation succeeds $\Leftrightarrow x = 0 \bmod 2$.
- So we recovered x_0.

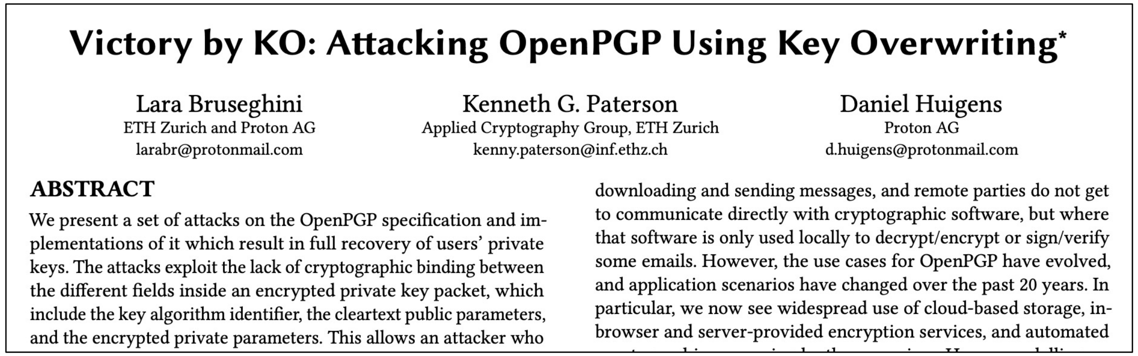
- Set g^* of order 4 mod p^*, set $y^* = (g^*)^{x0} \bmod p^*$.
- Now only x mod 4 is relevant, and we already know x mod 2.
- $(g^*)^x = y^* \bmod p^* \Leftrightarrow (g^*)^{x0+2x1} = (g^*)^{x0} \bmod p^*$

  $\Leftrightarrow ((g^*)^2)^{x1} = 1 \bmod p^*$

  $\Leftrightarrow x_1 = 0 \bmod 2$.
- So key validation succeeds iff $x_1 = 0 \bmod 2$.
- We can recover one new bit of x per iteration.

Other Aspects

- Full analysis of KO and KOKV attacks against OpenPGP spec and major implementations.
- Further attacks for non-CRT RSA and for ElGamal encryption; fault-style attack against EdDSA.
- Analysis of the extent to which two apps based on OpenPGP are vulnerable (ProtonMail, FlowCrypt).
- Discussion of countermeasures:
  - Robust key validation.
  - Use AEAD to bind key metadata and public key (AD) to encrypted private key.
  - Now adopted in OpenPGP Crypto Refresh!


Victory by KO: Attacking OpenPGP Using Key Overwriting*

Lara Bruseghini
ETH Zurich and Proton AG
larabr@protonmail.com

Kenneth G. Paterson
Applied Cryptography Group, ETH Zurich
kenny.paterson@inf.ethz.ch

Daniel Huigens
Proton AG
d.huigens@protonmail.com

ABSTRACT

We present a set of attacks on the OpenPGP specification and implementations of it which result in full recovery of users' private keys. The attacks exploit the lack of cryptographic binding between the different fields inside an encrypted private key packet, which include the key algorithm identifier, the cleartext public parameters, and the encrypted private parameters. This allows an attacker who

downloading and sending messages, and remote parties do not get to communicate directly with cryptographic software, but where that software is only used locally to decrypt/encrypt or sign/verify some emails. However, the use cases for OpenPGP have evolved, and application scenarios have changed over the past 20 years. In particular, we now see widespread use of cloud-based storage, in-browser and server-provided encryption services, and automated

ACM CCS 2022 🏆
https://www.kopenpgp.com/

Overview

- Introducing Key Overwriting Attacks
- Easier: OpenPGP
- **Harder: MEGA**
- Wrap-up

MEGA

- MEGA – E2EE cloud storage and communication platform with 280M registered users, 1000 Petabytes+ of stored data.

- Very strong security claims, promising users that MEGA cannot access user data.

- Three recent research papers invalidate this claim, all using KO attacks….

MEGA Key Hierarchy

Each user has:

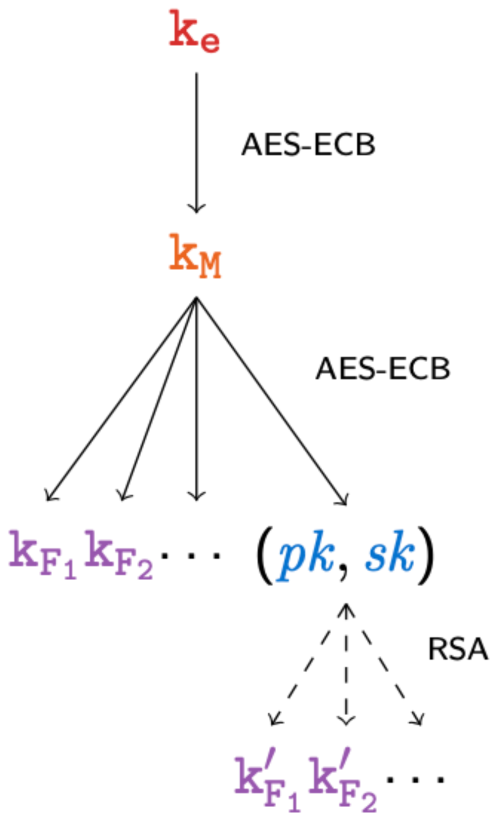
- a 128-bit *encryption key* k_e derived from password
- a 128-bit *master key* k_M
- a 2048-bit RSA keypair (pk, sk)
- *file encryption keys* k_{F1} , k_{F2} , . . .

The keys are encrypted using AES-ECB (!) and stored at server:

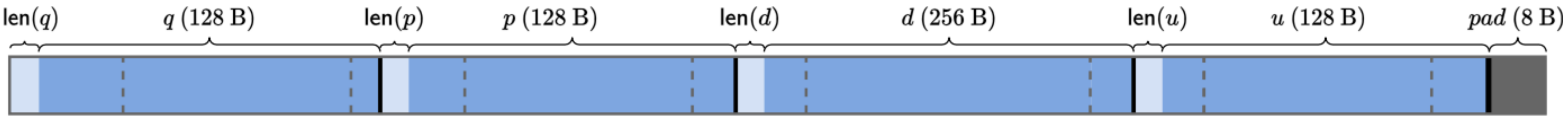
- $[k_M]_{ke}$
- $[k_F]_{kM}$, $[sk]_{kM}$

Shared-file encryption keys k'_F are encrypted under pk using RSA.

This RSA key is also used in the user login/authentication protocol.



MEGA RSA Private Key Format



- Custom encoding of *sk* for RSA-CRT decryption, referred to as privk:
  - the prime factors *p*, *q* of the RSA modulus,
  - the secret exponent *d*,
  - the value $u = q^{-1} \bmod p$.

- Each value is prefixed with a 2-byte length field.
- Split into 16-byte blocks for AES-ECB encryption with master key k_M.

MEGA Login Procedure

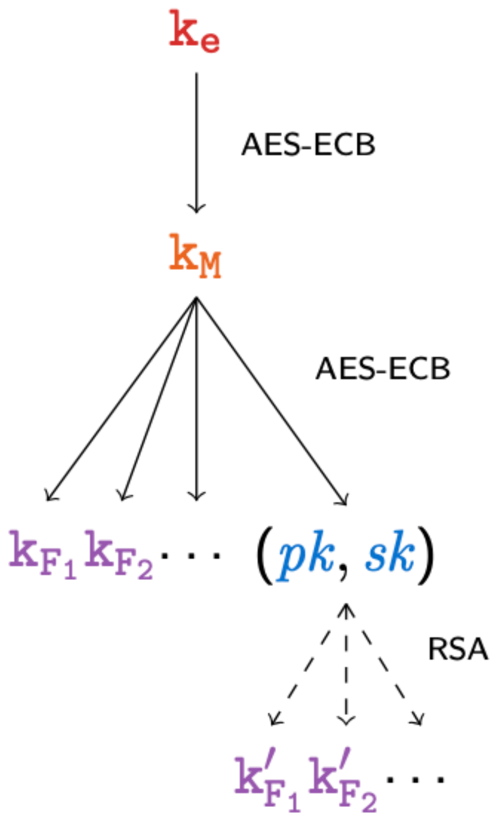
APPLIED CRYPTO GROUP

User		*MEGA*
	login request(*User*) →	
		get $([k_M]_{k_e}, [\mathtt{privk}]_{k_M}, \mathtt{uh})$ of *User*
		pick 43-byte $\mathtt{sid}$
		$[\mathtt{m}]_{pk} \leftarrow \mathsf{RSA.Enc}(pk, \mathtt{sid} \parallel \mathtt{uh})$
	← $([k_M]_{k_e}, [\mathtt{privk}]_{k_M}, [\mathtt{m}]_{pk}, \mathtt{uh})$	
$k_M \leftarrow \mathsf{AES\text{-}ECB.Dec}(k_e, [k_M]_{k_e})$		
$\mathtt{sid}' \leftarrow \mathbf{MegaDec}(k_M, [\mathtt{privk}]_{k_M}, [\mathtt{m}]_{pk}, \mathtt{uh})$		
	$\mathtt{sid}'$ or $\perp$ →	
		$\mathtt{sid}' \stackrel{?}{=} \mathtt{sid}$

ECB decryption using k_M to get privk, then RSA-CRT decryption and decoding, to recover sid´.

Attacking MEGA

- **Threat model:** malicious service provider trying to access customer data.
- **Goal:** obtain ECB decryption capability under k_M in order to recover *sk* (or any k_F).
- Attack cost measured mainly in the number of login attempts, since each login requires user interaction.
- What about KO attacks?
  - Use of ECB encryption means private keys are “somewhat malleable”.
  - Client always reconstructs public key from private key, so cannot overwrite public key directly.



Attacks Only Get Better…

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

IEEE S&P 2023
https://mega-awry.io/

512 logins

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

PKC 2023
eprint.iacr.org/2022/914

6 logins
(on unpatched version)

Caveat Implementor! Key Recovery Attacks on MEGA

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

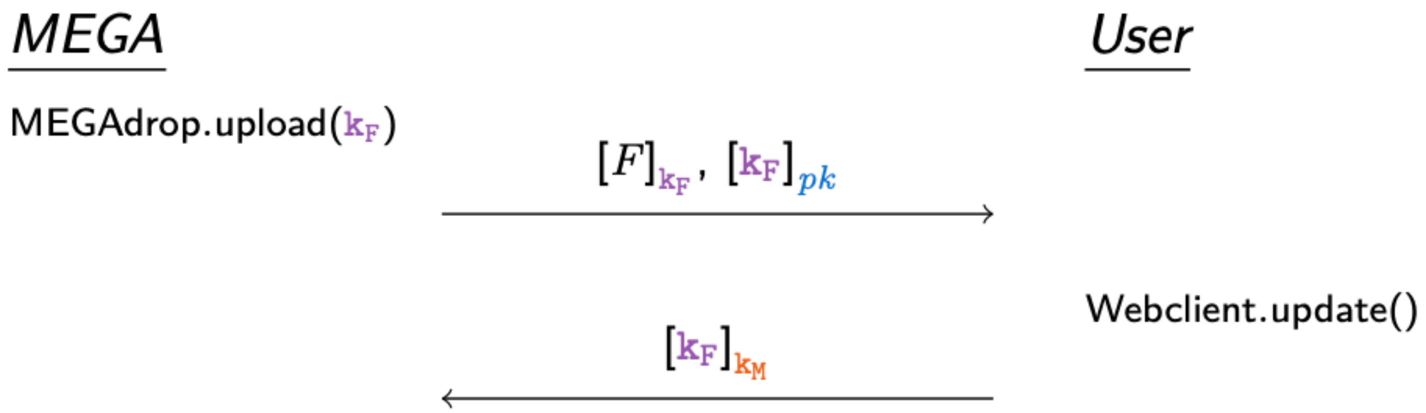
[1] King's College London
martin.albrecht@kcl.ac.uk
[2] Applied Cryptography Group, ETH Zurich
kenny.paterson@inf.ethz.ch, miro.haller@ethz.ch
[3] Information Security Group, Royal Holloway, University of London
lenka.marekova.2018@rhul.ac.uk

Eurocrypt 2023
mega-caveat.github.io/

2 logins
(on unpatched version)

Gadget 1: An ECB Encryption Oracle

- **MEGAdrop** allows anyone to upload files to a folder in the cloud storage of the recipient.
- Client automatically re-encrypts the received shared-file keys.



A malicious MEGA server can construct an ECB encryption oracle for k_M *without user interaction* and *without leaving any traces*!

Gadget 2: ECB Decryption Oracle from [BHP23]

APPLIED CRYPTO GROUP

User		*MEGA*
	login request(*User*) →	
		get ($[k_M]_{k_e}$, ~~$[\text{privk}]_{k_M}$~~, uh) of *User*
		pick 43-byte sid
		$[m]_{pk} \leftarrow$ ~~RSA.Enc(pk, sid ∥ uh)~~
	← ($[k_M]_{k_e}$, $[\text{privk}]_{k_M}$, $[m]_{pk}$, uh)	
$k_M \leftarrow$ AES-ECB.Dec(k_e, $[k_M]_{k_e}$)		
$\text{sid}' \leftarrow$ **MegaDec**(k_M, ~~$[\text{privk}]_{k_M}$~~, $[m]_{pk}$, uh)		
	sid' or $\perp$ →	
		~~$\text{sid}' \stackrel{?}{=} \text{sid}$~~

[q,p,d,u*]$_{kM}$ with u* containing two target ECB blocks

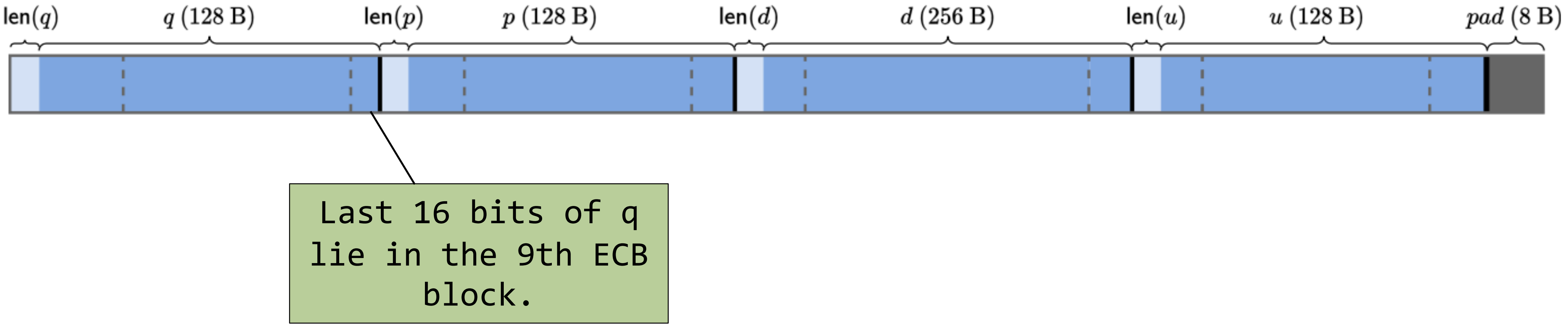
RSA.Enc(pk*,m*) with m* = u·q.

RSA-CRT decryption using u* in place of u → sid´ contains bytes of q·u*!

Post-process sid´ to recover two target plaintext blocks.

Combining the Gadgets

- Use ECB *encryption oracle* (Gadget 1) to overwrite $[\texttt{privk}]_{kM}$ with a completely known RSA private key (q*,p*,d*,u*).
- Use ECB *decryption oracle* (Gadget 2) four times to recover all 8 blocks (1024 bits) of the original q.
  - Would require 4 logins.
  - Does not quite work because of bad block alignments.
  - Guess and check for last 16 bits OR recover 9 blocks with 5 logins.



Combining the Gadgets

- Instead:

  - Recover last four 4 full blocks of q using just 2 logins → 512 bits of q.
  - For each guess for the 16 LSBs of q:
    - Use 512+16 = 528 LSBs of q in a lattice attack to try to recover all of q.
    - Check if q divides user's RSA modulus.

- Theory says this should work; our experiments show that it works just fine.

- Open problem: is there an attack on **unpatched** MEGA requiring only 1 login?

Patched MEGA client-side parsing and decryption

APPLIED CRYPTO GROUP

MegaDec(k_M, $[privk]_{k_M}$, $[m]_{pk}$, uh):

$sk \leftarrow$ DecryptPrivk(k_M, $[privk]_{k_M}$) // AES-ECB

$sid' \leftarrow$ DecryptSid(sk, $[m]_{pk}$) // RSA-CRT

Return sid'

Both steps perform extensive validity checking on the decrypted values and return distinguishable errors to the server!

Updated MEGA client-side parsing and decryption

Explicit errors due to validity checking:

- In DecryptSid(sk,·), a length check on the plaintext together with a legacy padding check reveal if the second byte of *m* is 0.
- Yields a novel "small subgroup meets Bleichenbacher" attack.

Implicit errors due to bugs in the low-level library:

- In DecryptPrivk(k_M, ·), a failure when recomputing $u \leftarrow q^{-1} \bmod p$ reveals if $\gcd(p, q) = 1$.
- Yields a novel attack based on modular inverses.

Both attacks involve **controlled** key overwriting of a victim user's RSA private key via ECB **encryption** oracle!

Attack Based on Modular Inverses

- Let $[B]_{kM}$ be the target ciphertext block.
- Let $\perp_{inv}$ be the error output by **MegaDec** if $\gcd(p, q) \neq 1$.
- Main idea is to use key overwriting to construct $[privk^*]_{kM}$ (using ECB enc oracle and $[B]_{kM}$) such that
  - $p^* \bmod r = 0$ for small prime r.
  - q^* contains B in the least-significant positions.
  - $q^* \bmod r$ has an attacker-controlled variable value.
  - e.g. $q^* = 2^{1024} + 2^{128+16} \cdot t + 2^{16} \cdot B + 1$ for attacker-selected t.

- Now $\perp_{inv}$ is output by **MegaDec** at client if and only if $q^* \bmod r = 0$, i.e.

$$\perp_{inv} \quad \text{iff} \quad 2^{1024} + 2^{128+16} \cdot t + 2^{16} \cdot B + 1 = 0 \bmod r. \qquad (1)$$

- Vary $t \bmod r$ across logins until $\perp_{inv}$ is output, and solve (1) to recover $B \bmod r$:

Attack Based on Modular Inverses

- So we can learn B mod r for small prime r in about r logins.
- Repeat for a set of primes r_i such that their product has 128 bits.
- We can then use CRT to learn B from the values of B mod r_i.
- Average cost: 627 login attempts and 66-91 ECB encryption oracle queries to recover one 128-bit block B.
- Run this attack 4 times to recover last 4 full blocks of original q, and apply lattice attack again.
- Recover RSA private key using approximately 2500 logins.
- So MEGA's updates did not improve security much!

Overview

- Introducing Key Overwriting Attacks
- Easier: OpenPGP
- Harder: MEGA
- **Wrap-up**

Armoury of Attack Vectors

- ECB mode
- Exotic + home-made encryption modes
- Lack of integrity mechanisms
- Improper use of integrity, e.g. MtE, E&M
- Padding oracle attacks
- Nonce reuse
- Lack of proper key separation/key reuse problems
- Bad interactions between different protocols
- Bespoke RSA padding schemes
- Roll-your-own authentication and key exchange protocols
- Naïve use of NaCl and other libraries
- Use of weak PRNGs or homebrew randomness generation methods
- Compression combined with encryption
- **Key Overwriting!**

Future Work

- Find other instances where KO/KOKV attacks apply.
  - Likely in badly-designed systems where key storage is outsourced to untrusted third parties.
- Study connections between KO/KOKV and fault attacks as per CHES community.
  - And to *tampering attacks* as studied in theoretical community.
- AEAD countermeasure seems clear.
  - Do we need a framework of formal security definitions and relations exploring security against KO/KOKV attacks?

- Thanks again to my co-authors:

  Martin Albrecht, Matilda Backendal, Lara Bruseghini, Miro Haller, Daniel Huigens, Lenka Mareková.

ETH zürich

Contact:

Professor Kenny Paterson
Applied Cryptography Group
kenny.paterson@inf.ethz.ch

ETH Zurich
Applied Cryptography Group
Department of Computer Science
Universitätstrasse 6
8092 Zurich, Switzerland

https://appliedcrypto.ethz.ch/

APPLIED CRYPTO GROUP