EPFL

March 2025

# Passive SSH Key Compromise via Lattices

Presented by Paul Tissot-Daguette, Léopold Galhaud, Roxanne Chevalley

CCS23

Keegan Ryan
Kaiwen He
George Arnold Sulliva
Nadia Heninger

EPFL

March 2025

# Plan

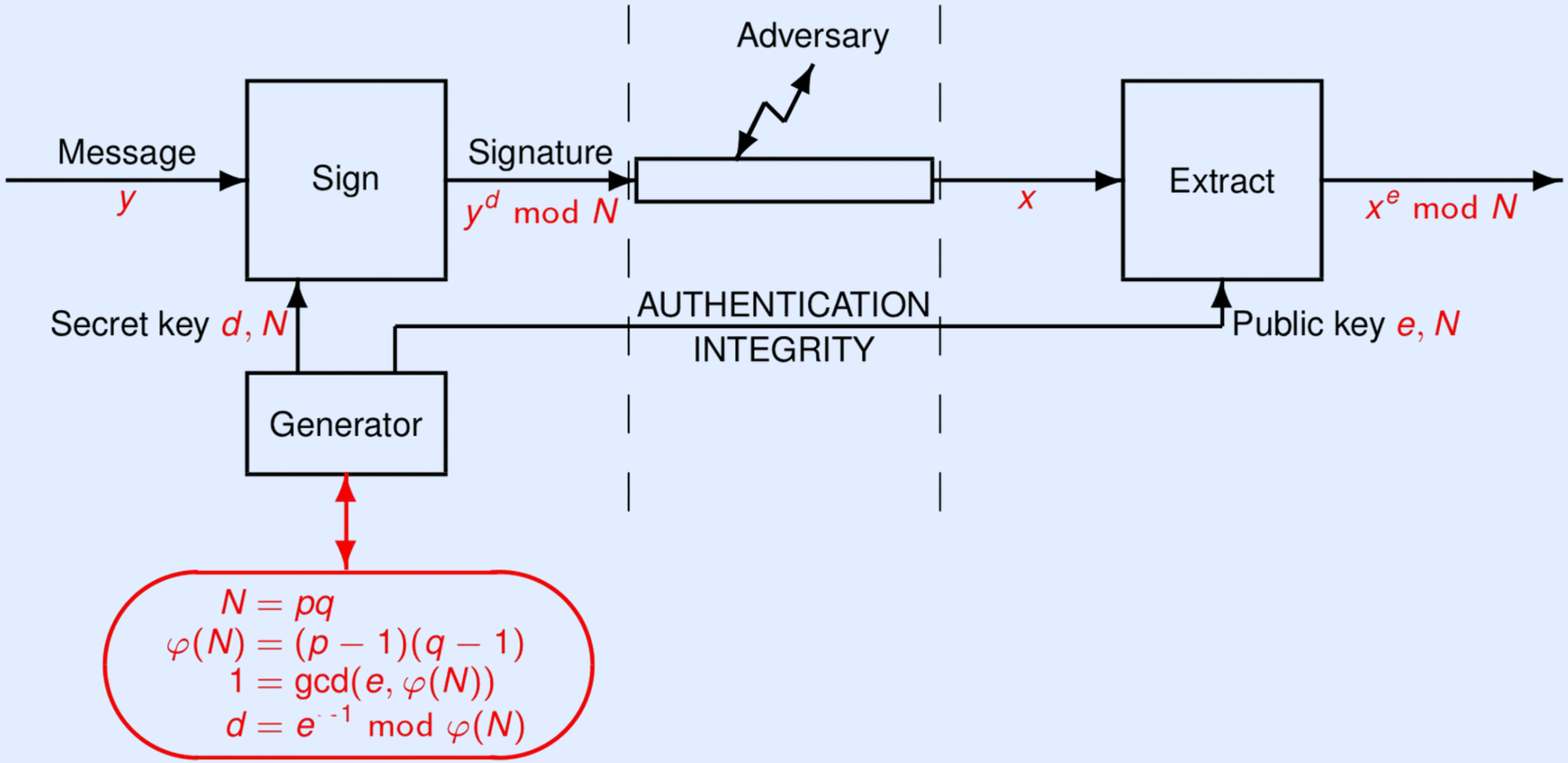
1. RSA signing scheme & fault attacks
2. Lattices & PACD
3. SSH protocol & security implications
4. Data collection & analysis

EPFL

March 2025

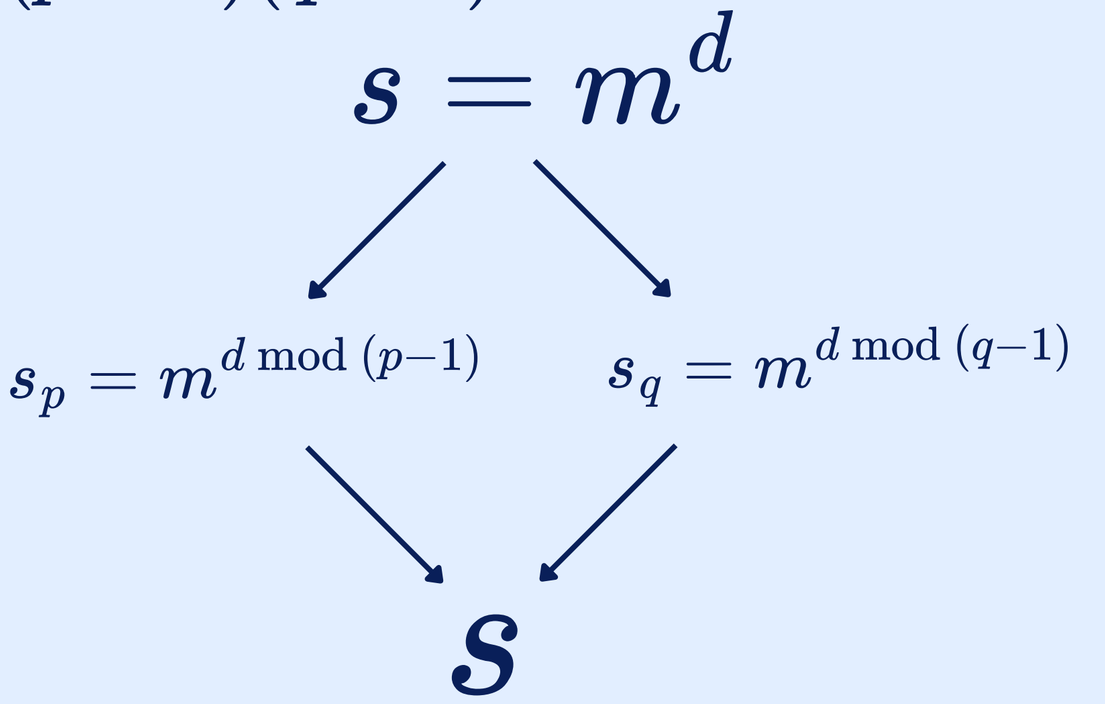
# RSA SIGNING SCHEME & FAULT ATTACKS

# Plain RSA Signature



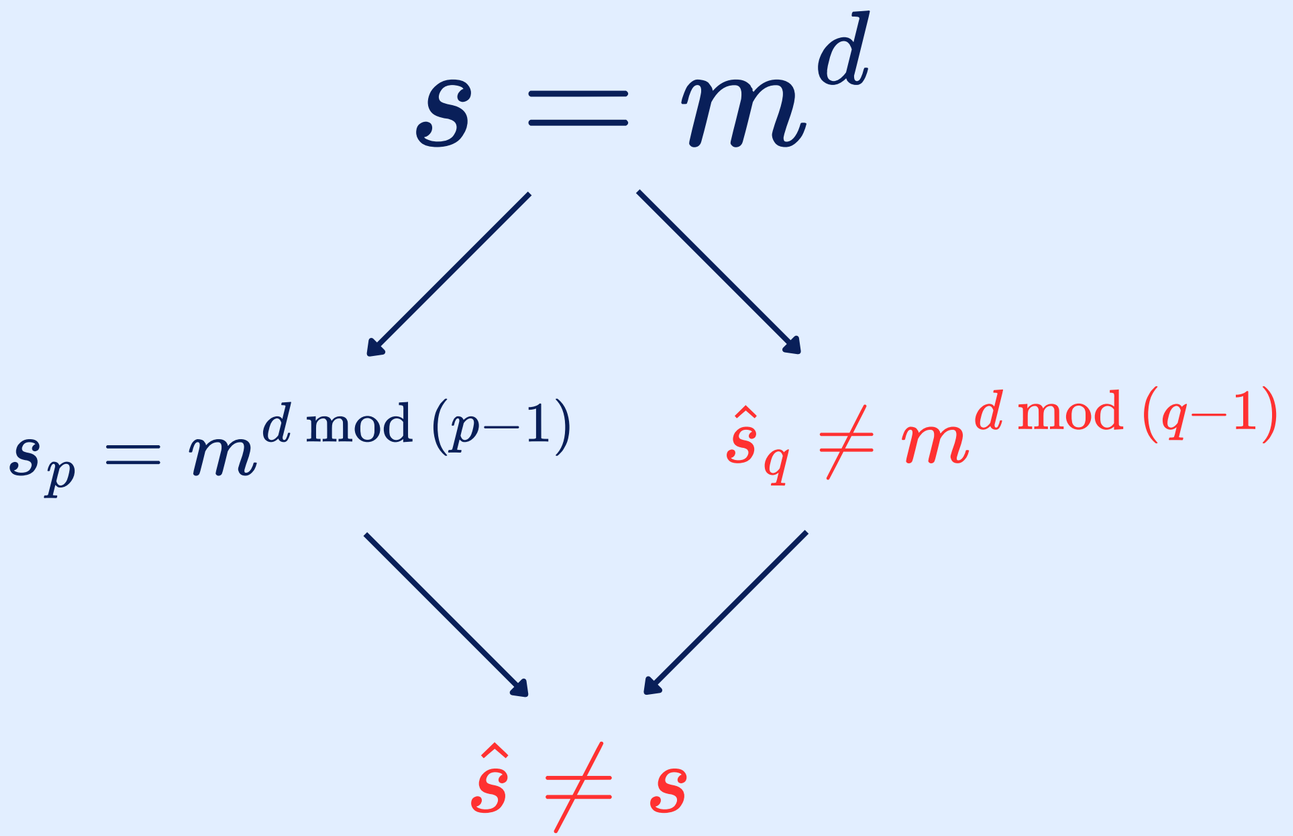
# CRT-RSA - a common optimisation

$$N = pq,\ \phi(N) = (p-1)(q-1)$$



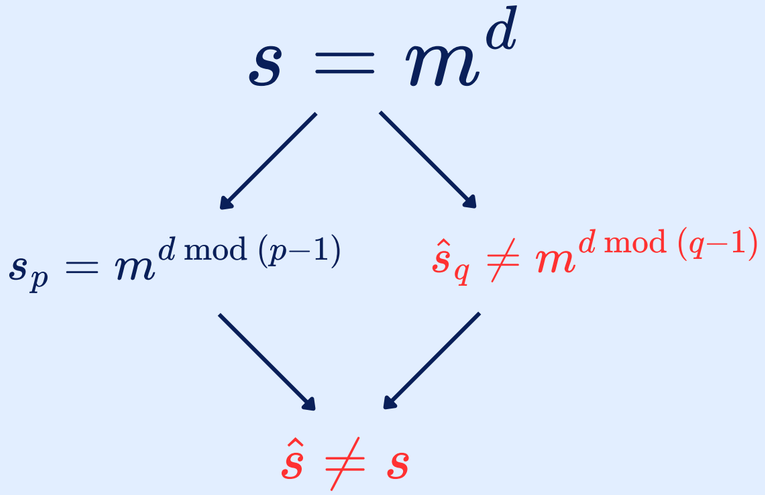
# RSA Fault Attacks

## The fault



# RSA Fault Attacks

## GCD attack on fully known messages



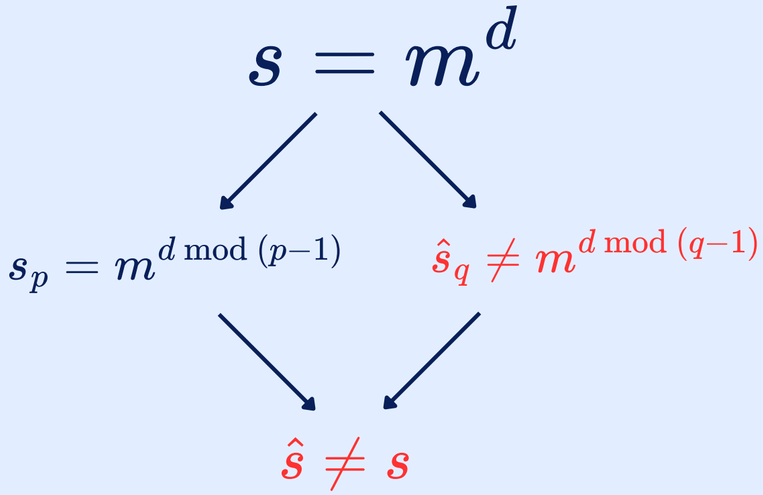

$$\hat{s} \equiv s \pmod{p} \implies \hat{s} - s = kp, k \in \mathbb{Z}^*$$

$$\hat{s} \not\equiv s \pmod{q} \implies \hat{s} \not\equiv s \pmod{N}$$

$$\gcd(N, \hat{s} - s) = p$$

# RSA Fault Attacks

## GCD attack on fully known messages




$$\hat{s}^e \equiv s^e \pmod{p} \implies \hat{s}^e - s^e = kp, k \in \mathbb{Z}^*$$

$$\hat{s}^e \not\equiv s^e \pmod{q} \implies \hat{s}^e \not\equiv s^e \pmod{N}$$

$$\gcd(N, \hat{s}^e - s^e) = \gcd(N, \hat{s}^e - m) = p$$

# RSA Padding - PKCS#1 v1.5

$$m = 00 \,||\, 01 \,||\, FF \ldots FF \,||\, 00 \,||\, ASN.1 \,||\, H(msg)$$

- ASN.1 - string identifying the hash function H
- H(msg) the message hash.
- Important property : determinism

# RSA Fault Attacks

## GCD attack on partially unknown messages

- Up until now, the attacker needed to know the message that was signed
- Coron et al. studied fault attacks against RSA signatures on partially unknown messages.
- m is in the range:

  [00 || 01 || F ... F || 00 || ASN.1|| 00..0, 00 || 01 || F ... F || 00 || ASN.1 || 11..1]
- midle point of range is a = 00 || 01 || F ... F || 00 || ASN.1|| 10..0
- Private key recovery possible when the output length of the hash function is at most 1/4 of the length of the public modulus

EPFL

March 2025

# LATTICE & PACD

# Background : Lattices

Discrete additive subgroup of $R^n$ specified by a basis:

Definition

$$\mathcal{L}(\vec{a}_1, \ldots, \vec{a}_n) = \left\{ \sum_{i=1}^{n} s_i \vec{a}_i;\ s_1, \ldots, s_n \in \mathbb{Z} \right\} = \{ A\vec{s};\ \vec{s} \in \mathbb{Z}^n \}$$

Determinant:

$$\det(\mathcal{L}) = |\det(A)|$$

# Attack on partially unknown messages

## Problem setup

- PCKS#1 v1.5 signature-padded message with an $l$-bits hash function:
  m = 00 || 01 || FF ... FF || 00 || ASN.1 || H(msg)

- We define *a* as the midpoint of the range of possible different values m can take:
  $m = a + y \quad \text{so} \quad |y| \leq 2^{\ell-1}$

# Attack on partially unknown messages

## Problem setup

$$m = a + y \quad \text{so} \quad |y| \leq 2^{\ell - 1}$$

$$\hat{s}^e = m = a + y \mod p$$

$$\hat{s}^e = kp + a + y \mod N$$

$$(\hat{s}^e - a \mod N) = kp + y$$

$$N = p \cdot q$$
$$k \in [1, q)$$

# PACD

## Partial Approximate Common Divisor

Let p be an unknown $\log p$-bit secret, and $N_0$, $N_1$ be $\log N$-bit samples of the form:

$$N_0 = pq_0$$

$$N_1 = pq_1 + r_1 \quad \text{for} \quad |r_1| \leq 2^{\log r}$$

The goal of the adversary is to recover p from $N_0$ and $N_1$

# Solving PACD

## Construction of our PACD instance

For a $b$-bit RSA public key $N = pq$, an $l$-bit hash function, and a single faulty signature $\hat{s}$ we construct the following PACD instance:

$$
\begin{aligned}
N_0 &= N = pq \\
N_1 &= (\hat{s}^e - a \mod N) = kp + y, \quad |y| \leq 2^{\log r}
\end{aligned}
\quad \Bigg| \quad
\begin{aligned}
N_0 &= pq_0 \\
N_1 &= pq_1 + r_1, \quad |r_1| \leq 2^{\log r}
\end{aligned}
$$

$$
\log N = b, \quad \log p = \frac{b}{2}, \quad \text{and} \quad \log r = \ell - 1
$$

# Solving PACD

## Part 1

$$Q_j(x) = N_0^{\max(k-j,0)} f(x)^{\min(j,k)} x^{\max(j-k,0)} \quad \text{for} \quad 0 \leq j \leq t$$

$$f(x) = N_1 - x$$

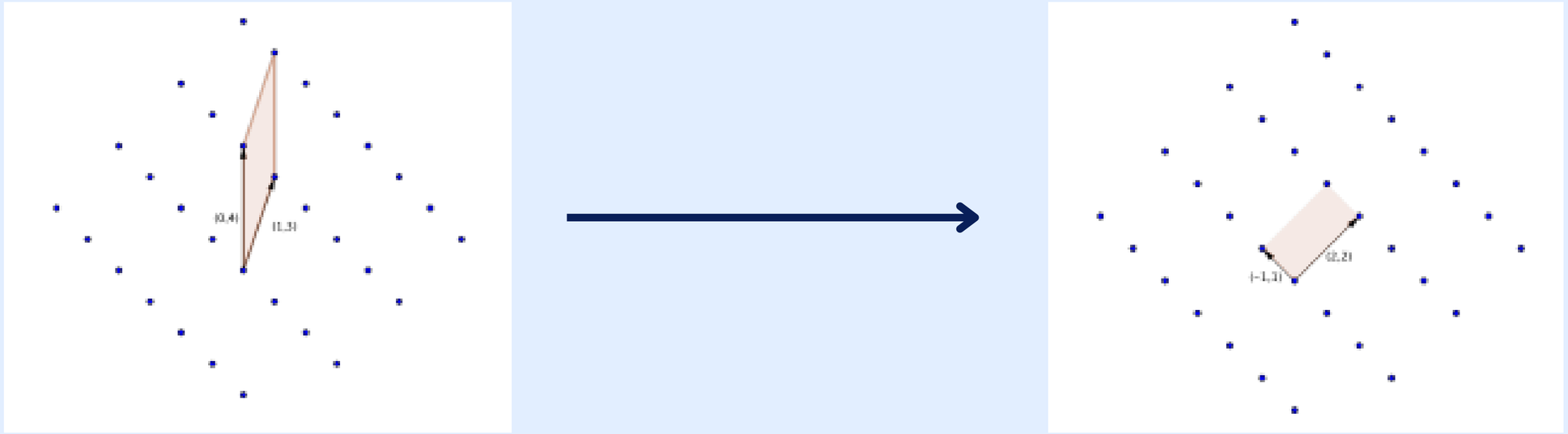
We construct a lattice where the basis vectors are the coefficient vectors of the polynomial $Q_j\left(2^{\log r} x\right)$ for parameters $(t, k) = (2, 1)$

$$B = \begin{bmatrix} -2^{2\log r} & 2^{\log r} N_1 & 0 \\ 0 & -2^{\log r} & N_1 \\ 0 & 0 & N_0 \end{bmatrix}$$

# Background : LLL

## Lenstra-Lenstra-Lovász Algorithm

Takes an arbitrary basis of a lattice and produces a reduced basis where the vectors are shorter and more orthogonal.



# Solving PACD

Part 2

$$B = \begin{bmatrix} -2^{2\log r} & 2^{\log r} N_1 & 0 \\ 0 & -2^{\log r} & N_1 \\ 0 & 0 & N_0 \end{bmatrix}$$

$$Q_j(x) = N_0^{\max(k-j,0)} f(x)^{\min(j,k)} x^{\max(j-k,0)} \quad y \text{ is a root} \pmod{p^k}$$

$$f(x) = N_1 - x \quad y \text{ is a root} \pmod{p}$$

- We use the LLL algorithm to find a reduced lattice
- Find a short vector and interpret it as the coefficients of a new polynomial: $g\left(2^{\log_r x}\right)$.
- We know that this new polynomial has the same root as $Q_j(x)$
- So if we have g a little enough polynomial such that $|g(y)| < p^k$ with $|y| \leq 2^{\log r}$

$$y \text{ is a root of } g\left(2^{\log_r x}\right). \quad \Rightarrow \text{we can find it !!}$$

# Solving PACD

Part 3

$$B = \begin{bmatrix} -2^{2\log r} & 2^{\log r} N_1 & 0 \\ 0 & -2^{\log r} & N_1 \\ 0 & 0 & N_0 \end{bmatrix}$$

In order to recover a suitable polynomial our lattice has to satisfy this condition:

$$\sqrt{\dim(B)}\, 2^{\dim(B)/4} \det(B)^{1/\dim(B)} < 2^{\log pk}$$

$$\dim(B) = t + 1$$

$$\det(B) = 2^{\frac{t(t+1)}{2}} \log r \, N_0^{\frac{k(k+1)}{2}}$$

If this bound is satisfied we can then recover a suitable polynomial $g$

# Solving PACD

## Our case

- For RSA keys in SSH where $\log p = (\log N)/2$, recovery is therefore possible for $\log r < (\log N)/4$
- This corresponds to faulty signatures where the hash length is up to $1/4$ the RSA modulus length.

# Parameters Selection

## Goals & Setup

- We have the equation : $Q_j(x) = N_0^{\max(k-j,0)} f(x)^{\min(j,k)} x^{\max(j-k,0)} \quad \text{for} \quad 0 \leq j \leq t$
- The goal is to minimize t (the lattice dimension) and secondarily k (the size of the elements in the lattice) while keeping a high probability of the attack succeeding
- Implementation in Python and SageMath and the lattice reduction algorithm is implemented in C++
- The experiments were run on Intel Xeon E5-2699 v4 CPUs running at 2.20GHz

# Parameters Selection

The experiment

- The implementation reveals that $k = \lfloor t/2 \rfloor$ is the optimal choice for $k$
- That means that if we have $\log p = (\log N)/2$ then there is a minimal value of t such that the Coppersmith parameters $(t, \lfloor t/2 \rfloor)$ succed with high probability.

# Parameter Selection

Average running time for $\log n = 1024$ for different values of $\log r$

| $\log r$ | $(t, k)$ | Dimension | Entry size (bits) | Avg. Time (s) |
|---|---|---|---|---|
| 169 | (2,1) | 3 | 1193 | 0.51 |
| 203 | (4,2) | 5 | 2454 | 0.49 |
| 218 | (6,3) | 7 | 3726 | 0.56 |
| 226 | (8,4) | 9 | 5000 | 0.60 |
| 231 | (10,5) | 11 | 6275 | 0.99 |
| … | … | … | … | … |
| 247 | (32,16) | 33 | 20335 | 26.74 |
| 248 | (38,19) | 39 | 24167 | 29.24 |
| 249 | (44,22) | 45 | 28005 | 40.10 |
| 250 | (52,26) | 53 | 33123 | 69.18 |
| 251 | (66,33) | 67 | 42073 | 157.66 |
| 252 | (88,44) | 89 | 56141 | 496.44 |
| 253 | (134,67) | 135 | 85555 | 2787.66 |

# Parameter Selection

## Average running time for common SSH parameters

| Host key type | $(\log N, \log p, \log r, v)$ | $(t, k)$ | Time (s) |
|---|---|---|---|
| RSA-1024,SHA1 | (1024,512,159,0) | (2,1) | 0.131 |
| RSA-2048,SHA1 | (2048,1024,159,0) | (2,1) | 0.130 |
| RSA-3072,SHA1 | (3072,1536,159,0) | (2,1) | 0.133 |
| RSA-4096,SHA1 | (4096,2048,159,0) | (2,1) | 0.135 |
| RSA-1024,SHA256 | (1024,512,249,6) | (44,22) | 835.219 |
| RSA-2048,SHA256 | (2048,1024,255,0) | (2,1) | 0.130 |
| RSA-3072,SHA256 | (3072,1536,255,0) | (2,1) | 0.133 |
| RSA-4096,SHA256 | (4096,2048,255,0) | (2,1) | 0.134 |
| RSA-1024,SHA512 | - | - | - |
| RSA-2048,SHA512 | (2048,1024,505,6) | (86,43) | 35485.211 |
| RSA-3072,SHA512 | (3072,1536,511,0) | (4,2) | 0.156 |
| RSA-4096,SHA512 | (4096,2048,511,0) | (2,1) | 0.171 |

EPFL

March 2025

# SSH PROTOCOL & SECURITY IMPLICATIONS

# SSH

- Protocol that creates a secure protected channel between a client and a remote server machine.
- Used for
  - Running commands remotely
  - Port fowarding
  - File transfer in SFTP and SCP

# SSH Handshake and server authentication

**Handshake**

1. Cipher negotiation
2. Diffie Hellman Key Exchange
3. **Server authenticates itself by signing SessionID* with its host key.**
4. Client verifies the public host key fingerprint.
5. From here, all messages are encrypted with keys generated from shared secret.
6. Client authentication happens after the initial SSH handshake, inside the encrypted tunnel.


**The passive adversary sees (unencrypted)**

- Cipher used
- Diffie hellman pre-keys
- Host's signature over the handshake Diffie-Hellman shared secret

They are unable to recover the shared secret

***SessionID = Hash(Client ID, Server ID, Client Cipher Message, Server Cipher Message, Shared Secret)**

# Security implications of compromised keys

An adversary steals a host's private signing key. What then ?

- **Doesn’t give the adversary the ability to decrypt passively collected SSH connections to the compromised host.**

- **Can however be leveraged in an active attack to impersonate the the compromised host.**
  - Pretend to be the host to a client, and steal their password.
  - If using PKA, the adversary can mimic server interaction to steal sensitive data.
  - If SSH Agent forwarding is enabled, the client accepts to serve as a signing oracle, which allows for full man in the middle.

March 2025

# DATA COLLECTION & ANALYSIS

# Data Collection

**Active Internet Scans** - Weekly IPv4 scans for 10 months

In one scan

- Around 22 million host with port 22 open
- Can perform 16 millions handshakes
- Get access to 3 to 5 million RSA host key signatures

**Historical scan data**

- Made available by Censys and University of Michigan.
- Lacks some metadata like cipher offering, SSH configuration, signature hashes.
- Scans starting April 2018 lack hash used to validate server signature.
- The lattice attack works even if we don't have the signature hashes, but we need it for the GCD attack.

# Results (1/2)

**Number of recovered keys**

- **1,250,000,000** SSH RSA host key signature.
- **593,671 (0,048%)** failed to validate.
- Out of this, **4,962** enabled recovery of the corresponding RSA private key with the lattice attack.
- Amounted to **189** unique RSA key pairs.

# Results (2/2)

**Affected devices :** five unique SSH version strings produced signatures resulting in factored keys.

| Host's SSH Version | Faults | Recent Host Count |
|---|---|---|
| SSH-2.0-Zyxel SSH server | **4705** | 3373 |
| SSH-1.99-Zyxel SSH server | 168 | 36 |
| *SSH-2.0-SSHD* | 87 | 11880 |
| *SSH-2.0-Mocana SSH 5.3.1* | 1 | 224 |
| *SSH-1.99-Cisco-1.25* | 1 | 83920 |

Zyxel servers almost never generate non-faulty signatures → permanent hardware fault.

**Current Zyxel devices are no longer affected because they use OpenSSL.**

EPFL

# Discussion

Countermeasure very simple

- Check that signature is valid before sending it
- Implemented in OpenSSL since 2001

Attack not critical as only a small number of legacy devices are affected.

Not much new research in the paper, but interesting discussion topics.

# Summary

- Seemingly benign computation errors in cryptographic protocol can lead to dangerous attacks.
  - In our case, one faulty RSA signature with the associated message leaks the SSH host key, using to the lattice-based attack.
- Can lead to full server impersonation.
- Modern devices and software implementations are mostly unaffected, but it’s important to keep the issue in mind.