Passive SSH Key Compromise via Lattices Seminar Report

Roxanne Chevalley
roxanne.chevalley@epfl.ch

Léopold Galhaud
leopold.galhaud@epfl.ch

Paul Tissot-Daguette
paul.tissot-daguette@epfl.ch

March 13, 2025

Abstract

The paper [5] demonstrates that a passive network attacker can extract private RSA host keys from an SSH server if a fault happens during the signature process. They use their method to identify hundreds of compromised keys in the wild from multiple vulnerable implementations.

1 Introduction

RSA digital signatures may expose the signer's private key if a computational or hardware error occurs during the signing process. In unprotected implementations that use the Chinese Remainder Theorem along with a deterministic padding scheme such as PKCS#1 v1.5, just one faulty signature combined with the public key and a single GCD computation is enough to recover the secret key. Previous studies identified that faulty hardware generating invalid signatures could expose RSA private keys, primarily exploited against TLS. This paper expands on that, showing that passive attackers can use lattice-based attacks (specifically a technique by Coron et al.) to recover RSA keys from single faulty PKCS#1 v1.5 signatures in SSH and IPsec as well.

The authors conducted large-scale internet scans, analyzing historical and current data, uncovering multiple vulnerable implementations due to hardware flaws. Their dataset included 5.2 billion SSH records, revealing nearly 600,000 invalid RSA signatures, leading to recovery of 189 unique RSA private keys.

2 Background

2.1 RSA Signature

RSA signatures are fundamental cryptographic primitives widely used in network protocols such as SSH, IPsec, and TLS. The textbook RSA signature for a message m involves raising it to the power of a private exponent d modulo a public modulus N: $s = m^d \mod N$. To set up RSA, we start by choosing two distinct large primes p and q and computing their product as the modulus: $N = pq$. Then, we select a public exponent e, coprime with the Euler's totient function $\varphi(N) = (p-1)(q-1)$. The private exponent d is computed as the modular inverse of e modulo $\varphi(N)$, $d = e^{-1} \mod \varphi(N)$. The security of RSA lies in the fact that without the knowledge of p and q, $\varphi(N)$ is hard to find. Thus it is a hard problem to recover d. Signature verification is performed by the recipient using the public exponent e: $m \stackrel{?}{=} s^e \mod N$. This verification works correctly because of Euler's theorem, which guarantees that $m^{ed} \equiv m \mod N$. This follows from the fact that $ed \equiv 1 \mod \varphi(N)$ implies $ed = 1 + k\varphi(N)$, and thus $m^{ed} = m^{1+k\varphi(N)} \equiv m \mod N$.

However, directly signing m ("textbook RSA") is insecure. Practical implementations therefore utilize padding schemes such as PKCS#1 v1.5, which deterministically hashes the message m with a hash function H and pads the result to match the length of N. Thus, the signature is computed as $s = \mathrm{Pad}(H(m))^d \mod N$. Due to the deterministic nature of PKCS#1 v1.5 padding, an observer knowing the message and hash function can reconstruct the exact padded message signed.

2.1.1 CRT-RSA Optimization

An optimization commonly applied to improve RSA signature computation performance is based on the Chinese Remainder Theorem (CRT), called CRT-RSA. CRT-RSA exploits the factorization of the modulus $N = pq$ to split one large modular exponentiation into two smaller, faster computations modulo p and q. First, compute two smaller exponentiations: $s_p = m^{d_p} \mod p$ and $s_q = m^{d_q} \mod q$, where $d_p = d \mod (p-1)$ and $d_q = d \mod (q-1)$. These intermediate results are combined using CRT as follows: $s = (s_p \cdot q \cdot (q^{-1} \mod p) + s_q \cdot p \cdot (p^{-1} \mod q)) \mod N$.

The CRT optimization significantly accelerates RSA signature computation because exponentiation modulo smaller primes p and q is substantially faster than exponentiation modulo their product N.

2.1.2 RSA Fault Attacks

RSA signatures using CRT-RSA optimization are susceptible to fault-based attacks, particularly when faults occur during modular exponentiation. A common scenario involves an error occurring during computation modulo one of the primes, say q. We compute $s_p = m^{d_p} \mod p$ and $\hat{s}_q \neq m^{d_q} \mod q$. This generates an invalid signature $\hat{s}$, which remains correct modulo the other prime p. Given a correct signature s and a faulty signature $\hat{s}$, an attacker can compute $\gcd(N, \hat{s} - s) = p$. This attack was later extended, indeed, knowledge of the signed message alone suffices, allowing the attacker to compute $\gcd(N, \hat{s}^e - m) = p$. In both cases, the knowledge of p is enough to get the factorization of N and thus recover the private key used to sign messages.

In an article called *Fault Attacks on RSA Signatures with Partially Unknown Messages* Coron et al. [1] introduced a more generalized lattice-based approach for fault attacks when the signed message is partially unknown. Specifically, in scenarios where PKCS#1 v1.5 padding is used, the padded message can be expressed as a linear expression $a+x$, where a is known and x is bounded by the hash function's output length. The faulty signature satisfies the linear congruence $\hat{s}^e = a+x \mod p$, which can be solved using lattice-based Coppersmith techniques.

2.2 SSH

SSH (Secure Shell) is a protocol that creates a cryptographically protected channel between a client and a remote server machine.

It is typically used for running commands on remote hosts and transferring files securely.

2.2.1 Handshake and server authentication

In order to establish the secure connection, the client and the server go through the following steps.

1. Agree on a cipher suite.
2. Perform a Diffie-Hellman key exchange to establish a shared secret.
3. Both parties compute the session identifier, a hash of their ids, the shared secret and the messages they sent to each other.
4. The server authenticates itself by signing the session ID with its host key.
5. The client verifies the signature to authenticate the server.

As soon as these steps completed successfully, all messages sent between the two parties are fully encrypted. As a result, a passive adversary observing the network can access the following information:

1. Cipher used.
2. Diffie-Hellman pre-keys.
3. Host's signature over the session identifier.

2.2.2 Implications of compromised host key

An attacker successfully stealing a server's private host key will be able to perform the following attacks

1. Impersonate the server to steal user's password, potentially leads to full man-in-the-middle.
2. If public key authentication is ued, it is possible to impersonate the server to steal user's private data, but full man-in-the-middle not directly possible.
3. With SSH Agent Forwaring is enabled on the client, then full man-in-the-middle is possible.

3 Lattice attack

3.1 Lattices

A lattice is a discrete additive subgroup of R^n specified by a set of vectors called a basis. A lattice is made of all the points that can be obtained thanks to linear combinations of the basis vectors. The main problem related to lattices that is useful in this work is the relatively short vector problem. In order to solve this problem we use the Lenstra-Lenstra-Lovasz algorithm (LLL) [3]. This algorithm reduces a lattice basis to smaller basis. This can be used in order to find a relatively short vector.

3.2 PACD

The Partially Approximate Common Divisor problem is a mathematical problem where the adversary has access to two integers $N_0 = pq_0$ and $N_1 = pq_1 + r_1$ with $|r_1| \leq 2^{\log r}$ and must recover p. This problem, first formulated by Howgrave-Graham [2], is solvable thanks to lattice-based algorithms.

The method to solve this problem used in this work [5] comes from May [4]. The first thing to see here is that if we can find r_1 then we can trivially compute $N_1 - r_1 = pq_1$ and then simply compute $\text{GCD}(N_1 - r_1, N_0) = p$. Therefore our goal is to find the value of r_1. It consists of building a polynomial $f(x)$ that has r_1 as a root mod p and to use this polynomial to build another set of polynomial $Q_j(x)$ depending on two variables t and k with $0 \leq j \leq t$. All the polynomials that can be constructed with different values of j all have r_1 as a root mod p^k.

We can now take the coefficients of the polynomial $Q_j(2^{\log r}x)$ and plug them into a matrix B. We then use this matrix as the basis of a lattice L. We can now run the LLL algorithm [3] to reduce the size of the basis of this L. The goal of this is to find a relatively short vector $\vec{v}$ that can then be interpreted as the coefficients vector of a new polynomial $g(2^{\log r}x)$. If the coefficients are small enough, then we can bind $|g(z)| \leq p^k$ for $|z| \leq 2^{\log r}$ and use $g(r_1) = 0 \mod p^k$ to conclude that $g(r_1) = 0$. We can simply find the roots of g and recover r_1 this way. As explained earlier this would lead us to a state where we can trivially compute p.

Now we should look into the conditions for this attack to succeed. For that we need to be able to recover a polynomial g small enough to satisfy the condition $|g(z)| \leq p^k$ for $|z| \leq 2^{\log r}$. In order for that to be the case, the lattice L has to satisfy the condition : $\sqrt{\dim(B)}\, 2^{\dim(B)/4} \det(B)^{1/\dim(B)} < 2^{\log p^k}$ with B the basis matrix of L, $\dim(B) = t + 1$ and $\det(B) = 2^{\frac{t(t+1)}{2}\log r} N_0^{\frac{k(k+1)}{2}}$. If this is satisfied we can then recover r_1 in polynomial time as explained in Ryan et al. [5]. What this bounds means in our RSA case is that it is possible to recover a short vector $\vec{v}$ and recover the root if the the hash length is up to a fourth of the RSA modulus length, which corresponds to : $\log r \leq \log N/4$

3.3 Partially unknown messages attack

When we only partially know the message, the situation is a bit more complicated. The method presented by Ryan et al.[5] is inspired from Coron et al. [1] and uses the PACD solving method presented in the paragraph above.

The first step is to represent a padded message m as a string made of known and unknown bits. We have i known bits as we already know the formatting of the PKCS#1 v1.5 protocol and the padding function utilized in our case. The rest of the message, the hashed message is unknown. If the original message was hashed with an l-bits hash function, we obtain $m = a+h$ with a being the midpoint of the range of the possible padded messages m and h satisfying $|h| \leq 2^{\log l-1}$. What this means is that we can obtain every possible padded message m by adding a to some value h that satisfies this bound.

This helps us set-up our PACD instance. From this form of m we can compute $(\hat{s}^e - a \mod N) = kp + h$ using $\hat{s}^e = m$ and the fact that the signature was incorrectly computed $\mod q$. Once we have this form we setup our PACD instance as such : $N_0 = N = pq$ and $N_1 = (\hat{s}^e - a \mod N) = kp + h$.

Now we can use the solving method presented earlier to find h and then we compute $\gcd(N_0, N_1 - h)$ to obtain p. We have successfully retrieved the private key !

3.4 The attack in practice

Now that we understood how the attack works, it is time to look into how efficient is it in practice. The two main vectors of optimization are minimizing the lattice dimension (minimizing t) and minimizing the size of the elements in the lattice (minimizing k) while making sure we keep a high probability of success for our attack. The attack was implemented using a mix of Python and SageMath and used a C++ implementation of the lattice reduction algorithm and was run on Intel Xeon E5-2699 v4 CPUs single cores running at 2.20GHz.

Empirical results show that the optimal value of k is $\lfloor t/2 \rfloor$. What this means is that there exists a minimal value of t such that parameters$(t, \lfloor t/2 \rfloor)$ succeed with high probability.

The main results from Ryan et al. [5] are that :

- A lattice of dimension 3 is sufficient to solve a PACD problem for relatively large errors
- The closer we get from the theoretical bound of $\log r = \log N/4$ the more the attack time increase. This can be problematic as some parameters used in SSH such as RSA-1024 with SHA-256 get close to it. In order to fix that and avoid an attack that takes too long, Ryan et al. [5] use a brute-force approach. They randomly guess the first bits of the unknown part of the message and try the attacks on the rest of the unknown bits. They try this until the lattice attack succeeds, that means that the guess was right. The computational cost of this attack is smaller than the cost of reducing very large lattices.

With most of the parameters combination often used in SSH, the attack time is very short ($< 0.2s$). There are two outliers. First of all RSA-2048 with SHA-512 takes way longer to compute as it is really close to the theoretical bound and requires a high dimension lattice to solve. Conveniently a very little amount of signatures used this combination of parameters in the dataset. Secondly, among the commonly used parameters combination, there is one the attack can not be run on : RSA-1024, SHA-512. Indeed here we are way over the theoretical bound of $\log N/4$ making the attack impossible.

4 Data Collection and Analysis

This section summarizes the authors' approach to identifying vulnerable signatures, detailing their methodologies and findings obtained through extensive active and historical data scans, SSH key recovery procedures, and thorough analyses of impacted devices.

Active Internet-Wide Scans Weekly IPv4 scans were conducted for 10 months. Each scan revealed:

- Around 22 million hosts with port 22 open.
- 16 million successful handshakes.
- 3 to 5 million RSA host key signatures observed.

Historical Scan Data Censys provided historical scan data, but it lacked metadata such as cipher offerings, SSH configurations, and signature hashes. The lattice attack does not require signature hashes, but they are necessary for the GCD attack.

Active SSH Key Recovery and Analysis Parsing the collected data resulted in 1,248,108,063 SSH RSA host key signatures. Of these:

- 593,671 (0.048%) RSA signatures failed validation.
- 4,962 enabled RSA private key recovery via the lattice attack.
- Recovered 189 unique RSA key pairs.

Processing invalid signatures required approximately 2080 core hours, while performing the lattice attack took 26 core hours.

Details of Affected Devices Only five unique SSH version strings produced signatures leading to factored keys. The most prevalent was the Zyxel SSH server.

Longevity of Stolen Keys Some hosts vanished from scans after generating faulty signatures. The longest-lived faulty key remained for 7 years, with a median of 4 months.

Fault Classification Zyxel servers producing faulty signatures almost never generated non-faulty ones, indicating a likely permanent hardware issue, probably in ZyWALL firewall devices. Modern Zyxel devices use OpenSSL and are unaffected. SSHD faults were intermittent, possibly linked to Hillstone Networks. Cisco and Mocana errors were rare, and a mitigation was deployed in 2022.

Long-Term Trends RSA usage is declining, whereas elliptic curves are becoming increasingly popular. Nevertheless, RSA-1024 remains consistently present. Although RSA-3072 was uncommon before 2020, its use has risen since 2022, aligning with the default settings introduced in OpenSSH 8.0 in April 2019. The majority of faulty signatures were observed between 2017 and 2020.

IPSec The paper also discusses IPSec and IKEv1-2, but no faulty signatures were found and no key was recovered.

5 Conclusion

The countermeasure against the attacked described in the paper is trivial. It suffices to validate signatures before sending them. OpenSSL implemented these protections since 2001. OpenSSH uses OpenSSL to generate signatures and is thus protected from this attack.

Lessons from these vulnerabilities highlight essential cryptographic protocol design principles, notably encrypting handshake data as soon as possible and validating critical operations before sending data over the network. Moreover, this demonstrates once again that the use of legacy devices can be risky in unexpected ways.

References

[1] Jean-Sebastien Coron et al. *Fault Attacks on RSA Signatures with Partially Unknown Messages*. Cryptology ePrint Archive, Paper 2009/309. 2009. URL: https://eprint.iacr.org/2009/309.

[2] Nick Howgrave-Graham. "Approximate integer common divisors". In: *International cryptography and lattices conference*. Springer. 2001, pp. 51–66.

[3] Arjen K Lenstra, Hendrik Willem Lenstra, and László Lovász. "Factoring polynomials with rational coefficients". In: (1982).

[4] Alexander May. "Using LLL-reduction for solving RSA and factorization problems". In: *The LLL algorithm: survey and applications*. Springer, 2009, pp. 315–348.

[5] Keegan Ryan et al. "Passive SSH Key Compromise via Lattices". In: *CCS '23* (2023), pp. 2886–2900. DOI: 10.1145/3576915.3616629. URL: https://doi.org/10.1145/3576915.3616629.