Passive SSH Key Compromise via Lattices

Keegan Ryan
kryan@ucsd.edu
University of California, San Diego
La Jolla, California, USA

Kaiwen He
khe01@mit.edu
University of California, San Diego
La Jolla, California, USA
Massachusetts Institute of Technology
Cambridge, Massachusetts, USA

George Arnold Sullivan
gsulliva@ucsd.edu
University of California, San Diego
La Jolla, California, USA

Nadia Heninger
nadiah@cs.ucsd.edu
University of California, San Diego
La Jolla, California, USA

ABSTRACT

We demonstrate that a passive network attacker can opportunistically obtain private RSA host keys from an SSH server that experiences a naturally arising fault during signature computation. In prior work, this was not believed to be possible for the SSH protocol because the signature included information like the shared Diffie-Hellman secret that would not be available to a passive network observer. We show that for the signature parameters commonly in use for SSH, there is an efficient lattice attack to recover the private key in case of a signature fault. We provide a security analysis of the SSH, IKEv1, and IKEv2 protocols in this scenario, and use our attack to discover hundreds of compromised keys in the wild from several independently vulnerable implementations.

CCS CONCEPTS

• **Security and privacy → Cryptanalysis and other attacks**.

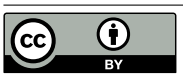
KEYWORDS

RSA, cryptanalysis, lattices

ACM Reference Format:
Keegan Ryan, Kaiwen He, George Arnold Sullivan, and Nadia Heninger. 2023. Passive SSH Key Compromise via Lattices. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS '23), November 26–30, 2023, Copenhagen, Denmark.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3576915.3616629

1 INTRODUCTION

RSA digital signatures can reveal a signer's secret key if a computational or hardware fault occurs during signing with an unprotected implementation using the Chinese Remainder Theorem and a deterministic padding scheme like PKCS#1 v1.5. This attack (due to Boneh, Lipton, and DeMillo [11] and Lenstra [35]) requires only a single faulty signature, the public key, and a single GCD computation, and it has been exploited extensively in the cryptographic side channel literature on active fault attacks.



CCS '23, November 26–30, 2023, Copenhagen, Denmark

ACM ISBN 979-8-4007-0050-7/23/11.
https://doi.org/10.1145/3576915.3616629

In a 2015 technical report, Weimer observed that this same vulnerability could be exploited in the context of TLS by an attacker without physical access to a machine, simply by connecting to machines and waiting for a fault to occur during computation [54]. He traced several of the failures he observed to failing hardware.

In 2022, Sullivan et al. [50] observed that this flaw remained exploitable on the open internet, and used passive network measurement to compute TLS private keys from vulnerable implementations that appeared to experience hardware failure.

However, this vulnerability was not believed to be realistically exploitable in the context of other popular network protocols like IPsec or SSH because the signature hash includes a Diffie-Hellman shared secret that a passive eavesdropper would be unable to compute, thus ruling out the single-signature GCD attack [54]. Because a passive adversary can typically collect significantly more data than an active adversary who must participate in every Diffie-Hellman exchange, this belief represented a significant underestimate of the cryptanalytic capabilities of such passive adversaries against SSH and IPsec compared to TLSv1.2.

In this paper, we show that passive RSA key recovery from a single PKCS#1 v1.5-padded faulty signature is possible in the SSH and IPsec protocols using a lattice attack described by Coron et al. [16]. In this context, a passive adversary can quietly monitor legitimate connections without risking detection until they observe a faulty signature that exposes the private key. The attacker can then actively and undetectably impersonate the compromised host to intercept sensitive data. We cast the key recovery problem as a variant of the partial approximate common divisor problem, and we show that this problem is efficient to solve for the key sizes and hash functions used for SSH and IPsec. For parameter settings that are near the asymptotic limits of these algorithms, we show how to balance the lattice attack parameters with an optimal amount of brute forcing to produce feasible running times.

We then carry out internet-wide scans for SSH and IPsec to measure the prevalence of vulnerable signatures in the wild. We find multiple vulnerable implementations that appear to be due to different classes of hardware flaws. We also carry out a retrospective analysis of historical SSH scan data collected over the course of seven years, and find that these invalid signatures and vulnerable devices are surprisingly common over time. Our combined dataset of around 5.2 billion SSH records contained more than 590,000 invalid RSA signatures. We used our lattice attack to find that more

than 4,900 revealed the factorization of the corresponding RSA public key, giving us the private keys to 189 unique RSA public keys. We also analyze passively collected SSH network data.

In addition to the signature vulnerabilities we were searching for, our analysis gives us a window into the state of the SSH, IKEv1, and IKEv2 deployment landscape. We observed a number of vulnerable and non-conformant behaviors among IPsec hosts in particular.

1.1 Contributions

- We observe that RSA host key signatures as used in the SSH protocol are vulnerable to a rarely applied lattice-based key recovery fault attack.
- We extensively analyze the feasibility and optimize this attack for the RSA key sizes and hash functions we observe in the wild.
- We identify several vulnerable SSH implementations.
- We also give current and historical measurements of SSH and IPsec hosts on the visible IPv4 space.

1.2 Ethics

We carried out our active network scans following best practices for network scanning. In particular, we identified ourselves and the purpose of the scan and provided an opt-out mechanism. We did not use compromised keys or forge protocol authentication. Since our active scans are connecting to visible hosts on the open internet, they do not involve any personally identifiable information that would merit approval or exemption by our institutional review board (IRB).

Our passive data collection came from two network taps at the University of California, San Diego. The location of the taps and our procedures for protecting sensitive information were developed with assistance from our IRB and our networking systems and security offices. Our data collection infrastructure and methodology has been deemed to be exempt by our IRB.

We took a number of steps to protect end users. We anonymized both IP addresses and MAC addresses for passively collected network traffic before storing it. Also, since we are observing SSH traffic, all content payloads are encrypted, and we merely parsed the initial SSH handshake data that set up the secure channel. Finally, the network tap machines and the data storage machines have restricted access both in terms of users and machines that have access and require multifactor authentication to access them.

1.3 Disclosure

Our research identified four manufacturers of devices susceptible to this key recovery attack. We disclosed the issue to Cisco on February 7, 2023 and to Zyxel on March 1, 2023. Both teams investigated promptly, although limitations in the historical scan data made it challenging to identify the software versions that generated the vulnerable signatures and reproduce the issue. Cisco concluded that Cisco ASA and FTD Software had introduced a suitable mitigation in 2022, and was investigating mitigations for Cisco IOS and IOS XE Software. Zyxel concluded that the issue had affected ZLD firmware versions V3.30 and earlier, which have been end-of-life for years. By the time of our disclosure, the ZLD firmware had begun using OpenSSL, which mitigates this issue.

Our attempts to contact Hillstone Networks and Mocana were unsuccessful, and we submitted the issue to the CERT Coordination Center on May 18, 2023 for assistance with disclosure. We received no additional information from CERT/CC during the 45-day disclosure period.

We considered notifying operators of affected devices whose keys we had recovered, but we determined this would be infeasible. Even after combining publicly available data with the historical scanning data, we were unable to determine which organization was responsible for a particular device, whether the device was still in use today, or up-to-date contact information for the device's current operator. We also lacked practical and actionable advice for owners of affected devices until the manufacturers completed their investigation.

2 BACKGROUND AND RELATED WORK

There is a decade-long history of related work in applied cryptography that examines large datasets from internet-connected hosts for vulnerable implementations [9, 12, 19, 27, 41, 51, 52, 54].

2.1 RSA Signing

An RSA public key consists of a public exponent e and a modulus $N = pq$ that is the product of two primes. The private key consists of the private exponent $d = e^{-1} \bmod \phi(N)$ and N.

A textbook RSA signature on a message m is the value $s = m^d \bmod N$. To verify the signature, a user checks if $s^e \bmod N = m$.

This naïve textbook approach is insecure [10], so in practice a padding function f is applied to m before signing. The signature of m is then $s = f(m)^d$.

2.1.1 PKCS#1 v1.5 Padding. The most commonly used RSA padding scheme in the protocols we consider is PKCS#1 v1.5, which hashes the message and then deterministically pads the resulting output to match the length of the RSA modulus. PKCS#1 v1.5 is described in RFC 2313 [31]. For a message m, the padded message is given by `pad(m) = 00 || 01||FF ... FF || 00 || ASN.1 || H(m)`. The value `ASN.1` is a string identifying the hash function H and `H(m)` is the message hash. Since the padding is deterministic, an observer who knows m and H can recover the full padded message.

RSA PKCS#1 v1.5 signature padding has been proven secure in the random oracle model [30]. The PSS and FDH padding schemes [5] are also sometimes used with RSA signatures, but they are not used in the context of SSH or IPsec. Unlike with PKCS#1 v1.5 padding, uncertainty in the message leads to uncertainty in most or all of the bytes of the padded message. Because of this, it is unlikely that the lattice attacks in this work extend to these other padding schemes.

2.1.2 CRT-RSA. A common RSA optimization is to use the Chinese Remainder Theorem (CRT) for modular exponentiation. Given private key values $d_p = d \bmod (p-1)$ and $d_q = d \bmod (q-1)$, the signature can be computed in parts $s_p = m^{d_p} \bmod p$ and $s_q = m^{d_q} \bmod q$, and then the complete signature, $s \bmod N$, can be reconstructed using the CRT.

2.2 RSA Fault Attacks

If a signing implementation using CRT-RSA has a fault during signature computation, an attacker who observes this signature

may be able to compute the signer's private key. These attacks exploit the fact that if an error is made while computing modulo one prime, say q, then the resulting invalid signature $\hat{s}$ is equivalent to the correct signature modulo one prime factor p, but not q.

2.2.1 GCD attack on fully known messages. Boneh, DeMillo, and Lipton noted [11] that if an attacker had a correct signature s and an incorrect signature $\hat{s}$ of this form then the attacker could compute $\gcd(N, \hat{s} - s) = p$. In an extension to this attack, Lenstra [35] observed that a single incorrect signature was sufficient as long as the attacker also knows the message that was signed. In this case the attacker can compute $\gcd(N, \hat{s}^e - m) = p$ to factor N.

There has been extensive work in the side channel community applying this attack to situations in which an attacker had access to the hardware doing the signing. In this situation the attacker would induce faults in hardware to recover faulty signatures [2, 8, 33, 46]. Weissman et al. [55] demonstrated a remote, active RSA fault attack using Rowhammer to induce faults in the signature computation.

There are many other works that consider fault attacks on RSA [3, 4, 13, 22]. Many of these have stricter requirements on the exact nature of the fault, and it is unlikely that a spontaneous fault in a given implementation results in the correct attack conditions. We focus on attacks exploiting arbitrary corruptions in CRT-RSA shares, since these attacks work for a broad range of faults.

Spontaneous faults in TLS. In 2015, Weimer [54] found naturally occurring faults in remote servers could also produce observable invalid signatures, and was able to compute TLS private keys in the wild by scanning network hosts and applying Lenstra's GCD attack. He traced some faults to hardware failures in cryptographic accelerator hardware.

The attack takes advantage of the fact that in TLS versions 1.2 [18] and below, a server's RSA signature is sent in the clear along with all of the handshake values necessary to validate it (client and server random nonces and server Diffie-Hellman parameters). Thus, a fully passive network observer has all of the information necessary to carry out the GCD attack against an observed invalid signature. In 2022, Sullivan et al. [50] demonstrated that vulnerable implementations continued to exist in the wild, and demonstrated fully passive key recovery from spontaneous computational faults.

In the current version 1.3 of TLS [47], the handshake messages occurring after the initial Diffie-Hellman key exchange are encrypted, and thus the server certificate and handshake signature are not visible to a fully passive attacker. An attacker who wishes to exploit this flaw would need to carry out an active attack by connecting to a server themselves to collect signatures.

2.2.2 Lattice attacks on partially unknown messages. In order to carry out Lenstra's GCD attack described in Section 2.2.1, the attacker needs to know the message that was signed. Coron et al. [16] study fault attacks against RSA signatures where the message is partially unknown. They primarily consider the message encoding specified in the ISO/IEC 9796-2 standard, which allows them to express the padded message as $a + bx + cy$ where a, b, and c are known, and x and y are unknown and bounded. When a fault leads to an RSA signature $\hat{s}$ that is incorrectly computed modulo q but correctly computed modulo p, one has the equation

$$\hat{s}^e = a + bx + cy \bmod p.$$

This is a linear equation in two variables with a small root modulo an unknown divisor of public modulus N, which can be solved using the Coppersmith-based techniques of Herrmann and May [28]. These methods reveal the unknown divisor p and thus recover the RSA private key. Coron et al. describe practical key recovery experiments in this setting and variants where the padded message can be written as a linear expression with more than two unknowns. Han, Wei, and Liu [23] analyzed the algorithm of [16] under the specific context of ISO/IEC 9796-2 padding where only the least significant bits of the message are unknown.

In the full version of the paper of Coron et al. posted to ePrint [17], the authors observe that PKCS#1 v1.5-padded messages are linear expressions $a + x$ in a single unknown variable where the size of x is bounded by the output length of the hash function. They conclude that private key recovery from an invalid PKCS#1 v1.5-padded signature is theoretically possible when the output length of the hash function is at most 1/4 of the length of the public modulus, but they report no practical experiments in this setting. Our work identifies a real-world setting for this attack, explores their observation in greater detail, and details several experiments to contextualize the behavior of this attack in practice.

2.3 Lattices and lattice-based cryptanalysis

A lattice is a discrete additive subgroup of $\mathbb{R}^n$. Concretely, a lattice is specified by a basis matrix of values to some precision. The main computational problem we need to solve over lattices in this work is to compute a relatively short vector given such an input basis. The Lenstra Lenstra Lovasz (LLL) algorithm [36] can be used to compute a vector of length $|v| \leq 2^{(n-1)/4} \det L^{1/n}$ for a lattice L of dimension n in polynomial time; current fast implementations can easily run on lattices of hundreds or even a few thousand dimensions [34, 48].

Coppersmith gave a lattice-based algorithm for factoring an RSA modulus given at least half of the bits of one of the factors [14]. Howgrave-Graham reformulated this problem as the approximate common divisor problem [29].

2.4 SSH Protocol

The Secure Shell (SSH) Protocol creates a cryptographically protected channel between a client and a remote server machine. Common applications include running commands on remote machines, port forwarding, and file transfer via SFTP or SCP. SSHv2 is specified by a number of RFCs, but our research focuses on RFC 4253 [57], which specifies the SSH transport layer protocol and RFC 8332 [7], which specifies the use of RSA with SHA-2 for client authentication.

SSH handshake and server authentication. SSH servers are identified by their public host keys. An SSH handshake begins with a cipher negotiation in which the client and server mutually agree upon a set of cryptographic algorithms, followed by a Diffie-Hellman key exchange. At this point, both client and server are able to compute a value known as the session identifier, which is a hash over the client and server identifiers, client and server key exchange messages, and the shared Diffie-Hellman secret. The server authenticates itself to the client by signing the session identifier with its host key; the

client verifies the public host key fingerprint (hash) out of band and validates the signature with the host public key. Further messages are encrypted using the shared secret.

Client authentication. Client authentication to the server happens after the initial SSH handshake, inside of the encrypted channel. With password-based authentication, the client sends the password in plaintext inside of the encrypted channel. With public key or host-based authentication, the client generates a digital signature with its client authentication key and sends it inside of the encrypted channel; the signature is bound to the session because it includes the session identifier hash. Since these methods occur within the encrypted channel, and the contents of their messages are not visible to a passive observer, our focus in this paper is on the server authentication keys.

A passive adversary monitoring the SSH connection would able to observe the cipher algorithms negotiated by the host and client, the public Diffie-Hellman key shares, and the host's signature over the handshake Diffie-Hellman shared secret. Because such a passive adversary is cryptographically unable to compute the shared secret, they cannot compute the message signed by the host and fully validate the signature themselves.

Algorithm choices. A number of cryptographic algorithms can be used in the SSH protocol. The key exchange may use Diffie-Hellman (RFC 4253 [57]), Elliptic Curve Diffie-Hellman (RFC 5656 [49]), or RSA (RFC 4432 [25]). In our data set, (Elliptic Curve) Diffie-Hellman key exchanges are by far the most common For host keys used to sign the key exchange material, the SSH protocol supports DSA, RSA, ECDSA, and Ed25519 (RFCs 4253, 5656, 7479 [43]). Although ECDSA and Ed25519 signatures are more modern, RSA signatures are still commonly used. Our research examines and exploits SSH host keys that are used to create RSA signatures.

RSA-based host keys are identified by the identifiers `ssh-rsa`, `rsa-sha2-256`, and `rsa-sha2-512`. Keys marked with `ssh-rsa` use the PKCS#1 v1.5 padding scheme with the SHA-1 hash function (RFC 4253), and those marked with `rsa-sha2-256` or `rsa-sha2-512` use the PKCS#1 v1.5 padding scheme with the SHA-256 and SHA-512 hash functions, respectively (RFC 8332). The SSH protocol does not prescribe any particular length for RSA host keys. With the September 2021 release of software version 8.8, OpenSSH has disabled support for `ssh-rsa` keys by default out of concerns about the use of SHA-1, but it continues to support `rsa-sha2-256` and `rsa-sha2-512`.

2.4.1 Security implications of compromised host keys. If an adversary is able to recover a server's private host signing key, this does not give the adversary the ability to decrypt passively collected SSH connections to the compromised host, but an active attacker may still mount an active attack impersonating the host. As a legitimate client attempts to initiate an SSH connection to the host with a compromised key, an active adversary in a network man-in-the-middle (MITM) position can respond to the client, complete the key exchange and sign the handshake with the recovered key. To the client, the public key matches the out-of-band fingerprint, and the signature verifies, so the client establishes an encrypted channel with the adversary. What happens next depends on the specific configuration expected by the legitimate host and client.

Password authentication. If the legitimate server and client use the "password" authentication method (RFC 4252 [56]), the adversary can offer this authentication method to the client. The client, believing it is connected to the legitimate server, responds with the plaintext password inside of the encrypted channel. The adversary can then initiate a new SSH connection with the true server, present the plaintext password when challenged by the true server, and relay messages between the client and server. To the client, the responses it sees match what it expects from the legitimate server, but the adversary can inject its own commands, modify responses, or connect to the server at a later date under this configuration.

Public key authentication. If the server and client use "publickey" or "hostbased" authentication, the attacker cannot relay messages in the same way. With these authentication methods, the legitimate client generates a digital signature over a value that includes the session identifier of the connection it believes it has established. At this point, an active attacker could simply act as an endpoint and attempt to impersonate the server by receiving the client's messages and fabricating the server's responses. Such an adversary could accept files uploaded by the client via SFTP or SCP, or receive a password from a client's attempt to run commands as root.

However, this attack alone does not immediately allow a true man-in-the-middle attack without a further cryptographic vulnerability. This is because if the man-in-the-middle attempts to connect to the host it wishes to impersonate, it will not be able to simply relay the client's authentication signature to the legitimate server, because the session identifier in the signature hash is bound to the connection by including both Diffie-Hellman key shares and the negotiated secret.

SSH agent forwarding. If the client has enabled SSH agent forwarding, even more malicious actions are possible. Although disabled by default, SSH agent forwarding is frequently used in conjunction with public key authentication methods and allows a client to initiate SSH connections from one remote host to a second without the client's private keys leaving the client's local machine. This uses the SSH agent, a program containing the client's private keys that runs on the client machine and can be queried to request an authentication signature for a server the client has access to. When SSH agent forwarding is enabled, a UNIX-domain socket is created on the remote host and connected to the client's local SSH agent. Access to this socket via the remote server, and therefore to the SSH agent, can be used as a signing oracle when attempting to authenticate to the second server. In the context of our attack, the client connects to the adversarial host, believing it is legitimate, and creates the SSH agent forwarding socket. The adversary can then attempt to connect to legitimate hosts, using the forwarded agent as a signing oracle to pass public key authentication challenges as if it were the client. This scenario is particularly dangerous, as the final host compromised by the adversary may be totally unrelated to the host that first generated the vulnerable signature. OpenSSH 8.9 introduced mechanisms to restrict how keys in the local agent can be used and mitigate this type of attack in February 2022 [42], but adoption is limited.

2.5 IPsec protocol suite

IPsec is a protocol suite defined by RFC 2408 [39] (ISAKMP), RFC 2409 [24] (IKE), RFC 7296 [32] (IKEv2). IPsec aims to provide confidentiality, data integrity, and source authentication to traditional IP datagrams. IPsec is often used to implement Virtual Private Network (VPN) solutions. A crucial part of IPsec is the Internet Key Exchange (IKE) protocol, which performs cipher suite agreement, key exchange, and authentication between two hosts, called the initiator and responder.

There are two versions of the IKE protocol: IKEv1 and IKEv2. Although IKEv1 is a legacy protocol, both protocols are still commonly deployed. In both versions, the protocol begins by establishing a security association (SA) — agreeing on a set of cipher suites and performing an initial Diffie-Hellman key exchange. Then the two parties each authenticate themselves and the handshake. Both protocol versions offer a complex set of options for authentication. The details depend on the protocol version as well as the multiple symmetric and asymmetric authentication modes available in each version. The IKE handshake authenticates both the initiator and the responder. Which one authenticates first differs based on the version and mode.

2.5.1 IKEv1. The legacy IKEv1 protocol is the original version of IKE. There are multiple ways to authenticate an initiator in IKEv1: digital signatures, public key encryption, and pre-shared key (PSK). There are also two modes that can be combined with any authentication mode: Main mode and Aggressive mode. Aggressive mode is designed to exchange fewer messages.

A fully passive RSA fault attack is not possible in Main mode because all communications, including authentication, are encrypted between the initiator and responder once they have completed the key exchange. Thus in Main mode, a passive adversary would be able to see the public Diffie-Hellman values and the cipher suites, but the initiator and responder authentication is encrypted and thus would not be visible to a passive eavesdropper. An adversary would have to actively connect to a server to receive signatures.

However, a passive fault attack is possible in Aggressive mode. Aggressive mode reduces the number of round trips by doing cipher negotiation, key exchange, and authentication in a single round trip. In IKEv1 aggressive mode authenticated with digital signatures, both the initiator and responder signatures are sent in plaintext during the first round trip exchange, which is not yet encrypted. These digital signatures are visible to a passive adversary.

The relevant exchange in Aggressive mode is outlined below (From section 5.1 of RFC 2409 [24]):

```
Initiator: SA, KE, Ni, IDii
Responder: SA, KE, Nr, IDir, [ CERT, ] SIG_R
```

In this exchange, the initiator chooses the preferred cipher suites in the Security Association (SA) payload, and then put the Diffie-Hellman public values in the Key Exchange (KE) payload. It provides its identity (IDii) to the responder.

Our attacks require knowledge of both the signature and public key. The former is present in the responder's SIG payload, and the latter is in certificates transmitted by CERT payloads. Some responders are configured to only send a CERT payload if the initiator explicitly requests it by sending a Certificate Request (CERTREQ) payload in its first message, which could be sent in any message (RFC 2408).

The initiator and responder compute `HASH_I` and `HASH_R` respectively, which are hashes that depend on the Diffie-Hellman shared secret. These hashes are signed over these hashes to achieve mutual authentication. For a passive adversary that observes an IKEv1 Aggressive mode handshake, only the responder's signature is sent in plaintext, and it signs an unknown message.

Nonstandard signature padding. Interestingly, the format of the responder's SIG payload subtly differs from the standard PKCS#1 v1.5 padding for RSA signatures as described in Section 2.1.1. RFC 2409 includes the following comment about padding in signatures: "RSA signatures MUST be encoded as a private key encryption in PKCS #1 format and not as a signature in PKCS #1 format (which includes the OID of the hash algorithm)."

According to RFC 2313 [31] (PKCS#1 v1.5 padding) Section 8.1, for "private-key operations", the block type (second octet of padded message) is either `00` or `01`. RFC 2409's remark on using "encryption" rather than "signature" format appears to be intended to mean that the ASN.1 OID is *not* prepended to the hash unlike the standard PKCS#1 v1.5 padding. The apparent reason is that `HASH_R` is computed from a negotiated PRF so the algorithm does not need to be identified by an OID, but the full security implications of specifically excluding the OID from the signature is unclear.

In summary, the format of the padded message to be signed in the IKEv1's AUTH payload is

```
00 || 01 || FF ... FF || 00 || HASH_R
```

Security implications of compromised signing keys in IKEv1. An adversary who is able to compute the secret signing key for a host would be able to cryptographically impersonate that host (either initiator or responder) to another party by forging valid signatures during an IKEv1 handshake using digital signature authentication. However, because IKEv1 involves mutual signature-based authentication of the negotiated Diffie-Hellman secret by both the initiator and responder, a true man-in-the-middle attack seems difficult for such an adversary to carry out. Such an attack seems to require compromising both initiator and responder signing keys or exploiting a further cryptographic vulnerability.

2.5.2 IKEv2. IKEv2 was introduced in 2005 and is not backward compatible with IKEv1.

IKEv2 allows a wider variety of authentication modes than we have space to fully describe here. From the perspective of the security analysis in our paper, digital signatures in IKEv2 are always embedded in authentication payloads (AUTH), and in all authentication modes, the AUTH payload is sent in encrypted form after the establishment of a security association. Thus a passive adversary cannot observe signatures in IKEv2. In addition, for most modes, the initiator always authenticates before the responder.

The exception to this is the Extensible Authentication Protocol (EAP) authentication method in IKEv2. EAP is of interest to our work because an active adversary performing a scan for IKEv2 hosts does not need to authenticate itself as a legitimate client before it obtains a potentially faulty signature from the responder.

The EAP mode exchanges from [32] relevant to our data collection described in Section 4.4.2 are shown below.

```
Initiator: SAi1, KEi, Ni
Responder: SAr1, KEr, Nr
Initiator (encrypted): IDi, [CERTREQ,] SAi2, TSi, TSr
Responder (encrypted): IDr, [CERT,] AUTH, EAP
```

The first exchange is similar to that of IKEv1. The second exchange is encrypted using the shared Diffie-Hellman secret established during the first exchange. During the second exchange, the initiator sends an identity payload (IDi) and the responder responds with its own identity (IDr), an authentication payload (AUTH) that authenticates the prior IKE exchanges (including cipher suites and nonces) and IDr. Note that IKEv2 allows multiple ways to authenticate data in an AUTH payload. The most common one (and the one we are most interested in) is an RSA digital signature in PKCS#1 v1.5 format.

PSK authentication. In addition to using signatures, the responder might also authenticate using a MAC derived from a pre-shared key. For the responder, it is computed as follows [32, Section 2.15]:

```
AUTH = prf( prf(Shared Secret, "Key Pad for IKEv2"),
                    <ResponderSignedOctets>)
```

It is well known that this allows an offline dictionary attack if the the pre-shared key (`Shared secret`) has low entropy, like a user-chosen password. RFC 7296 specifically notes that "it is a common but typically insecure practice to have a shared key derived solely from a user-chosen password without incorporating another source of randomness". An adversary can obtain the MAC from the responder, brute force low-entropy passwords, and verify if the responder's MAC authenticates with the guessed payload.

2.5.3 Security implications of signing key compromise in IKEv2. In IKEv2 authentication, the authentication messages are bound to the connection by including all of the packet payloads. Thus an adversary who obtains the secret signing key for only one side of the connection may be able to cryptographically impersonate that host. If, as in IKEv1, both the IKEv2 initiator and responder mutually authenticate via signatures, a true man-in-the-middle attack may not be possible, as signatures are bound to the individual connection and cannot simply be forwarded. Unlike IKEv1, however, IKEv2 also allows for the signature-based authentication to be one-sided, meaning other attack models can be considered.

PSK authentication. A man-in-the-middle attack may be possible if one side is using pre-shared key authentication with a password that the adversary is able to guess. Unlike IKEv1, which specifies the same authentication algorithms for both initiator and responder, RFC 7296 states that "there is no requirement that the initiator and responder sign with the same cryptographic algorithms. The choice of cryptographic algorithms depends on the type of key each has. In particular, the initiator may be using a shared key while the responder may have a public signature key and certificate. It will commonly be the case (but it is not required) that, if a shared secret is used for authentication, the same key is used in both directions."

Thus in a key compromise scenario where an initiator is using pre-shared key authentication and the adversary has compromised the responder's signing key, the adversary may be able to forge authentication messages for both sides, and carry out a man-in-the-middle attack.

EAP. EAP authentication methods can allow a full man-in-the-middle attack if the initiator is authenticated with EAP, especially for EAP methods that do not establish a shared key. RFC 7296 specifically states that "EAP methods that do not establish a shared key SHOULD NOT be used, as they are subject to a number of man-in-the-middle attacks [1] if these EAP methods are used in other protocols that do not use a server-authenticated tunnel." If an adversary compromises the responder's key, then the IKEv2 encryption tunnel is no longer considered "server-authenticated", and the adversary can relay EAP messages as described in [1].

An example of a particularly vulnerable EAP method in the context of this attack scenario is EAP-MS-CHAPv2. EAP-MS-CHAPv2 is supported by a broad range of platforms (including Windows, macOS, and Android). If the initiator is authenticated with EAP-MS-CHAPv2, an attacker who carries out a man-in-the-middle attack will be able to obtain the full MD4 hash of the initiator's password due to an attack by Marlinspike et al. [38] with complexity 2^{56} (the security level of single DES). This is true regardless of the entropy content of the password. In MS-CHAPv2, the MD4 hash of the password is a password-equivalent: an attacker with the knowledge of the hash can successfully authenticate to the responder as the initiator. An attacker that obtains such a hash may then authenticate to the responder without the initiator's participation at all.

While the exact security implications of host key compromise in IKEv1 and IKEv2 depend on the particular configurations of the initiator and responder, the potential impact of host impersonation due to a faulty signature exposing an RSA signing key is large.

3 LATTICE ATTACK

In this section, we show how to use Coppersmith-like techniques to determine the factorization of the RSA modulus $N = pq$ from a faulty signature whose message is only partially known.

Our method, like that of Coron et al. [16], uses a lattice approach for solving linear equations modulo unknown divisors of integers. However, their work is specialized to the case of the ISO/IEC 9796-2 standard, in which there are two unknowns. They apply the approach of Herrmann and May [28] to this case.

In our specific setting of PKCS#1 v1.5 padding with unknown hash value, the linear equation only has a single unknown. Although Coron et al. observe that this easier problem should be solvable asymptotically, they do not explore this idea in depth. The restriction to a single unknown simplifies their original attack, and we remark how the problem of key recovery from a single faulty RSA signature is best understood as an instance of the partial approximate common divisor problem. In this section, we analyze this case in detail, providing theoretical recovery bounds and profiling the performance of the attack for the parameters used in SSH.

Problem setup. Recall that a PKCS#1 v1.5 signature-padded message, when interpreted as a big-endian integer, has the hexadecimal representation `0001ffffff...ff00yy...yyzz...zz`.

Here, `yy...yy` are the known digits from the `ASN.1` string identifying the hash function, and `zz...zz` are the unknown digits from the hash function output. For an ℓ-bit hash function, this means that the unknown padded message is represented by an integer m in a range of size 2^ℓ. Let a be the midpoint of this range and write

$m = a + h$ so $|h| \leq 2^{\ell-1}$. In a faulty signature $\hat{s}$ that is correctly computed modulo p, we have $\hat{s}^e = m = a + h \bmod p$.

Because $\hat{s}^e = a + h \bmod p$ and $\hat{s}^e \neq a + h \bmod q$, then

$$\hat{s}^e = kp + a + h \bmod N$$

for some $k \in [1, q)$. In other words, $(\hat{s}^e - a \bmod N) = kp + h$ is approximately a multiple of p and is a sample of the Partial Approximate Common Divisors (PACD) problem. This problem is often defined for arbitrarily many samples, but because faulty signatures are rare, we focus on the minimal case of knowing one noisy PACD sample. Here, log is the base-2 logarithm, rounded up to the nearest integer.

Definition 1 (Partial Approximate Common Divisors (PACD) [29]). *An instance of the PACD problem with two samples is parameterized by the bit lengths of the following inputs. Let p be an unknown $\log p$-bit secret, and N_0, N_1 be $\log N$-bit samples of the form*

$$N_0 = pq_0$$
$$N_1 = pq_1 + r_1 \text{ for } |r_1| \leq 2^{\log r}.$$

The goal of the adversary is to recover p from N_0, N_1.

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

$$N_0 = N = pq$$
$$N_1 = (\hat{s}^e - a \bmod N) = kp + h$$

with parameters $\log N = b$, $\log p = b/2$, and $\log r = \ell - 1$. Different combinations of modulus length and hash length lead to different parametrizations, and so different methods may be used to solve the PACD instance.

3.1 Solving PACD

There are a number of lattice constructions for solving the PACD problem and variants in the literature [53]. The simplest lattice constructions apply in the case when one has many noisy samples, and they predict that at least $\log N/(\log p - \log r)$ samples are needed. For our application, this would require collecting more than one faulty signature for each public key that has a fault in the same share of the CRT calculation. In order to solve this problem with a single sample, one can use a Coppersmith [15]/Howgrave-Graham [29]-type approach. May [40] gives a concise introduction to this method.

Concretely, this approach considers the function $f(x) = N_1 - x$ which has a small root modulo p at $x = r_1$. The approach is parameterized by integers (t, k) with $1 \leq k \leq t$, and considers the constructed polynomials

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

These polynomials, as well as any integer linear combination of the polynomials, have a small root modulo p^k at $x = r_1$.

As is standard in Coppersmith-style attacks, we construct the Euclidean lattice where the basis vectors are the coefficient vectors of $Q_j(2^{\log r}x)$. For the parameters $(t, k) = (2, 1)$, this is the lattice spanned by the rows of matrix

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

This lattice is reduced using the LLL algorithm to find a short vector $\vec{v}$ which is interpreted as the coefficients of polynomial $g(2^{\log r}x)$. If the coefficients are suitably small, we may bound $|g(y)| < p^k$ for $|y| \leq 2^{\log r}$ and use $g(r_1) = 0 \bmod p$ to conclude $g(r_1) = 0$. Finding the rational roots of g is tractable, allows recovery of r_1, and then $p = \gcd(N_0, N_1 - r_1)$.

With one PACD sample, [40] gives the sufficient condition

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

where $\dim(B) = t + 1$ and $\det(B) = 2^{\frac{t(t+1)}{2} \log r} N_0^{\frac{k(k+1)}{2}}$ for recovering a suitable polynomial g. The following theorem gives asymptotic bounds for recovering the root r_1 in polynomial time.

Theorem 3.1. *[40, Theorem 1, $\delta = 1$] Given positive integers N_0, N_1 and bounds $\log p/\log N \gg 1/\sqrt{\log N}$ and $2^{\log r}$, we can find r_1 such that*

$$\gcd(N_0, N_1 - r_1) \geq 2^{\log p}$$

and $|r_1| \leq 2^{\log r}$, provided that

$$2^{\log r} < c N_0^{(\log p/\log N)^2}.$$

The algorithm runs in polynomial time in c and $\log N_i$.

For RSA keys in SSH where $\log p = \log N/2$, recovery is therefore possible for $\log r < \log N/4$, which corresponds to faulty signatures where the hash length is up to 1/4 the RSA modulus length. This matches the bounds in [17], showing that representing faulty signatures as PACD samples is as effective as representing them as linear equations modulo unknown divisors.

These feasibility results are typically stated in terms of asymptotic polynomial-time bounds, but in practice smaller parameters suffice for successful recovery in practice. This translates to lattice bases of smaller dimension and entry size, and so our actual implementation differs from the described attack in a few ways.

3.2 Experimental Parameter Selection

We wish to determine the smallest parameters for which the lattice attack is expected to successfully recover the RSA private key from a faulty SSH PKCS#1 v1.5 signature. In particular, we primarily wish to minimize t (and therefore the lattice dimension) and secondarily wish to minimize k (and therefore the size of elements in the lattice basis) such that the attack succeeds with high probability.

We implemented the Coppersmith/Howgrave-Graham algorithm, as well as our reduction from the faulty signature problem to PACD, using a combination of Python and SageMath. We also used a fast lattice reduction implementation written in C++ that is capable of reducing lattice bases with large entries [48]. We ran our experiments on individual cores of Intel Xeon E5-2699 v4 CPUs running at 2.20GHz.

We create synthetic faulty signatures and perform the attack with a range of small parameters. We generate faulty signatures by generating a correct signature and corrupting one of the shares

Table 1: Minimal parameters achieving $\geq 99\%$ recovery rate in the lattice attack for an assortment of hash lengths ($\log r$). We set $\log N = 1024$ and $\log p = 512$. Running times are averaged over 100 randomly generated samples.

$\log r$	(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

modulo p or q uniformly at random. Our implementation empirically reveals that $k = \lfloor t/2 \rfloor$ is the optimal choice for k, as it is capable of successfully recovering the factorization for the largest error size $\log r$. To reduce the size of the search space, we fix k to this value in future runs. This means that for each problem instance with $\log p = \log N/2$, there is a minimal value of t such that the Coppersmith parameters $(t, \lfloor t/2 \rfloor)$ succeed with high probability. We report the results for $\log N = 1024$, corresponding to a 1024-bit RSA modulus, in Table 1.

Two features are immediately clear from the data. First, a small lattice of dimension 3 suffices to solve PACD for relatively large errors. Second, the running time of the attack increases dramatically as $\log r$ approaches the theoretical bound $\log N/4$. This becomes problematic when we attempt to run the attack for certain SSH parameters, such as RSA-1024 with the SHA-256 hash function. To mitigate this issue, we perform a hybrid attack where we brute force the leading bits of the error and attempt the lattice attack for each guess; if the lattice attack succeeds, we have guessed correctly. This makes the value of $\log r$ used in the lattice attack slightly smaller, but it makes the dimension of the lattice significantly smaller. The exponential cost of brute forcing a few bits is less expensive than the polynomial cost of reducing a large lattice. We introduce an aditional parameter v to represent the number of bits we attempt to brute force. We could have alternatively explored using a chaining approach as developed by Bi et al. [6], but signatures that required brute forcing bits and large dimension lattices were rare, and it was not worth optimizing this case further.

Table 2 lists the chosen attack parameters for common SSH configurations. We observe that this Coppersmith-style attack is extremely fast to recover an unknown 160-bit SHA-1 hash for any RSA modulus size, and so it is efficient to determine if a non-validating signature reveals the secret key. Although the cost of attacking a faulty RSA-2048 signature using SHA-512 is large, these signatures are rare in our dataset. The largest share these parameters had in our measurements was 6% (as 2048-bit keys constitute the majority of the SHA-512 use reported in Table 5); even with the strong assumption that this is representative of the internet as a whole, we estimate that a single one of our servers could comfortably process over 3000 invalid signatures per day. We observed that 3.2 billion total signatures contained approximately 590,000 invalid RSA signatures, so we estimate that a single unoptimized server could process tens of millions of passively observed SSH connections per day.

Table 2: Selected parameters and running time of the attack on various parameter sizes found in SSH host keys. We do not report parameters for `RSA-1024,SHA512` because the number of unknown bits in the hash is well beyond what we can brute force or solve with lattices. Overall, we see that the lattice attack is quite efficient, and the lattices (with dimension $t+1$) are usually quite small, enabling fast reduction. Running time of the attack is averaged over 100 trials, and our attacks had 100% success rate.

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

Among these realistic parameter settings, only the case of SHA-512 with a 1024-bit RSA modulus is infeasible for this algorithm to recover in practice, because a 512-bit hash length is well beyond the theoretical limit of $\log N/4$ bits.

4 DATA COLLECTION AND ANALYSIS

In order to search for vulnerable signatures, we collected network data from a number of sources, including active internet-wide scans, historical scan data, and passive network taps.

4.1 Active SSH Data Collection

We performed active SSH scans and supplemented our data with historical internet-wide scans made available by Censys [20].

4.1.1 Contemporary scans. For this paper, we carried out weekly scans of the IPv4 space between the dates of October 26, 2022 and August 22, 2023. We first used the ZMap [21] scanning tool to scan for hosts with port 22 open, and then used the ZGrab2 `ssh` module to perform a SSH handshake up to the server authentication step. With ZGrab2, we used the `--host-key-algorithms=ssh-rsa` command line argument to offer only `ssh-rsa` as an authentication option. ZGrab2 does not currently support `rsa-sha2-256` or `rsa-sha2-512` as host key algorithms, so these types of host keys are not represented in our scan data. While this is a limitation of

our data, our passive data collection reveals that `ssh-rsa` is more common than `rsa-sha2-256` and `rsa-sha2-512` combined, so this implies our contemporary scans still capture the majority of SSH hosts and can be compared with the historical scans, which also lacked these host key types. In these contemporary scans, ZGrab2 saves the hash of this message to be signed by the server host key.

In a typical scan, we saw around 22 million hosts with port 22 open, completed around 16 million SSH handshakes, and saw 3 to 5 million RSA host key signatures. Of those RSA signatures, around 3,000 failed to validate per scan.

4.1.2 Historical ZMap scan data. We supplemented our current scans with a collection of historical SSH internet-wide scan data collected at the University of Michigan and Censys at various time periods beginning in April 2015 and ending in June of 2020 made available by Censys. Although Censys still collects data on SSH, their updated scanner no longer collects signature data. These historical datasets were collected using various versions of ZMap and ZGrab, with a variety of different SSH configuration and cipher offerings, not all of which are documented.

Scans beginning April 3, 2018 included the hash used to validate the server signature; the others did not. While we ran the lattice attack on invalid signatures regardless of whether the hash was known or unknown, we were able to use the known server hashes to additionally attempt the GCD attack and investigate other reasons why some RSA signatures did not validate.

Figure 1 shows the proportion of host keys using RSA, ECDSA, or DSA algorithms. Over the course of our historical scan dataset, the number of hosts responding on port 22 rose from around 20 million hosts in 2015 to 26 million in 2020. This corresponds to approximately 0.6% of IPv4 space. The number of completed connections per scan increased from 10 million to 16 million, and 99.99% of the successful connections in our dataset included a public key and signature.

4.2 Active SSH Key Recovery and Analysis

4.2.1 Attempting RSA Key Recovery. In total, our combined data set of contemporary and historical SSH scan records consisted of 5,202,311,657 SSH records, of which 3,189,469,782 included the host public key and signature used during the SSH key exchange step. Of these 3.2 billion signatures, 1,248,108,063 (39.1%) were RSA. 593,671 (0.048%) of the RSA signatures failed to validate. Of the invalid signatures, 4,962 allowed us to recover the corresponding RSA private key using our lattice attack. This corresponded to 189 unique RSA key pairs, since many hosts either shared the same keys or generated multiple invalid signatures in the data. For the purposes of this section, a "valid" signature is correctly structured, an "invalid signature" fails validation for any reason, and a "faulty" signature enables factorization by the lattice method.

Our analysis was performed on a cluster of machines that included a combination of Intel Xeon E5-2699 v3 CPUs running at 2.30GHz and Intel Xeon E5-2699A v4 CPUs running at 2.40 GHz. Processing the raw connection data in search of invalid signatures consumed approximately 2080 core hours, and performing the lattice attack to recover keys from invalid signatures consumed 26 core hours.

Table 3: Vulnerable implementations. We observed version strings identifying four vendors among the SSH signatures that revealed private keys. We compare the prevalence of such signatures to the number of hosts in our August 22, 2023 scan with this version string, and observe different rates of vulnerable signatures across vendors. Per RFC 4253, the version number 1.99 indicates a server supports SSH-2, but also supports older SSH protocol versions.

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

4.2.2 Details of Affected Devices. Our analysis of active scan data revealed five unique SSH version strings that produced signatures resulting in factored keys, detailed in Table 3. The most prevalent software version is the "Zyxel SSH server" with 4873 vulnerable signatures followed by "SSHD" with 87. "Mocana SSH 5.3.1" and "Cisco-1.25" both generated a single vulnerable signature. We examined the behavior of hosts using these version strings over time. In particular, once a private key is exposed by a faulty signature, how long does the host continue to use that key? We also want to classify the nature of the errors; that is whether the error is permanent (all future signatures are faulty) or transient (the host generates valid signatures after the fault).

Our data do not perfectly reveal whether two signatures were created by the same physical device. A device may use multiple RSA keys, a single RSA public key could be used by multiple load-shared devices, and a single device may change its IP address between scans. Because a compromised key affects all devices that share the key, we make the assumption that a RSA public key uniquely identifies a host machine. Our results do not change significantly when we identify hosts by the pair of their public key and IP address.

We examined all signatures from hosts that presented an SSH version from the list of five in Table 3. Our goal is to understand the window of time in which the adversary can perform the attacks in Section 2.4.1 after compromising a private key, so we first looked at the length of time a compromised key continues to be used by the SSH host to sign messages. This does not capture the case of a client continuing to trust an old host key after the host rotates keys or goes offline, but we believe this metric is nevertheless useful for observing general trends. Several of the hosts did not reappear in our scans after generating a faulty signature, and the longest-living compromised host remained online for over seven years after the first collection of a vulnerable signature. The median duration of the attack window was around 4 months.

We partition the observed hosts into a number of classes. These include whether the host generates no faulty signatures, generates only faulty signatures, generates a non-faulty signature after generating a faulty signature (transient fault), and generates only faulty signatures after the first fault (permanent fault). The results of this classification are given in Table 4 and suggest that there are

Table 4: Classifications of Potentially Faulty Hosts. We identified hosts using their public key and classified host versions based on whether a host with a given key never, always, or intermittently produced signatures resulting in factored private keys over our dataset. We observe different behaviors across vendors, suggesting different underlying root causes for the vulnerable signatures.

Version	No Faults	Only Faults	Transient	Permanent
Zyxel	48172	133	3	3
SSHD	32117	0	48	0
Mocana	2231	0	1	0
Cisco	450681	0	1	0

different underlying root causes for signature faults in the types of host we observed.

Notably, the majority of Zyxel hosts that produced faulty signatures almost never produced non-faulty signatures in our scan data, suggesting some type of permanent hardware failure. All 48 of the SSHD hosts that generated a faulty signature eventually generated a non-faulty signature, and 14 of these produced more than one faulty signature, which suggests that the invalid signatures are produced by a process that may recover. Both of the Mocana and Cisco hosts generated non-faulty signatures after generating the faulty signature, and the invalid signatures were so rare that the error condition appears to be truly transient.

Zyxel manufactures a variety of networking equipment for both individual consumers and businesses. Based on manual fingerprinting of some of the affected hosts, it appears that the faulty signatures originated from Zyxel-manufactured ZyWALL firewall devices. Hastings et al. [26] and Sullivan et al. [50] both previously identified issues in RSA keys used by Zyxel devices. [26] factored RSA keys by observing that RNG errors led to RSA keys sharing a common secret factor that can be found by computing the GCD, and [50] observed Zyxel devices creating a faulty RSA signature as part of a TLS connection. In their response to our disclosure, Zyxel concluded that legacy ZLD firmware versions V3.30 and earlier may be affected, and currently available products defend against this by using OpenSSL.

It is more difficult to identify the faulty devices that present the "SSH-2.0-SSHD" version string. However, some of the IP addresses that generated the faulty signatures also had open ports responding to TLSv1.2 connections with certificates. These certificates included an organization name of "Hillstone Networks" and a common name of "SG-6000." Hillstone Networks manufactures networking equipment, and the SG-6000 is a rack-mounted firewall that performs network inspection and analysis. Hillstone Networks was also previously identified by Hastings et al. [26] and Sullivan et al [50] due to the presence of the same aforementioned vulnerabilities.

We have little information about the single Cisco host that generated a faulty signature. Cisco also manufactures networking equipment and had products that were also affected by the issues in [26] and [50]. According to Cisco, the SSH version string indicates that the signature was likely generated by Cisco IOS, IOS XE, ASA, or FTD Software. In 2022, Cisco ASA and FTD Software introduced a mitigation to prevent faulty RSA signatures in TLS[1] that also mitigates the issue we found with SSH, and they are evaluating mitigation strategies for Cisco IOS and IOS XE Software.

Little information is available about the Mocana host. Some of the other hosts that identified themselves by the same version string also hosted web pages for monitoring a HP 2530 network switch manufactured by Hewlett Packard. Hewlett Packard was also among the network equipment manufacturers affected by the issue in [26].

For all of these affected devices, we can conclude that they did not incorporate countermeasures against fault attacks, despite these countermeasures being well known for decades.

4.2.3 Long Term Trends. Our active scan data empirically reveal several interesting trends in the cryptographic methods used by SSH implementations. As time has passed, RSA has apparently become less common, and elliptic curve cryptography has become more common. This is shown in Figure 1. While there are other explanations for this trend, such as RSA hosts being firewalled from the public internet or `ssh-rsa` host keys being deprecated and replaced by `rsa-sha2-256` keys, we conclude that the increasing popularity of elliptic curve cryptography is the simplest explanation.

In addition, the sizes of RSA keys has shifted over time. This is shown in Figure 2. While RSA-2048 remains the most common key size in our data, RSA-1024 has maintained a consistent presence since the beginning of our scan data. The use of RSA-3072 was uncommon up through 2020, but it has a substantial share in our contemporary scans beginning 2022. Note that the RSA key generation implementation in OpenSSH 8.0 changed the default RSA bit size from 2048 to 3072 in April 2019, which may partially explain this change.

Finally, Figure 3 shows that vulnerable signatures are most prevalent in the historical scan data between 2017 and 2020. As the majority of the observed faulty signatures came from a single software version, the prevalence may simply reflect the introduction and eventual upgrading of this software.

4.2.4 Other causes of invalid signatures. We explored other possible errors that could have led to the signature being invalid. To help diagnose the cause, we considered the active scan records that included enough information to reconstruct the message hash. There were 457,944 such invalid signatures that we considered. Of these, 27,956 were invalid but had correct PKCS#1 v1.5 signature padding. Of the invalid signatures with correct PKCS#1 v1.5 padding, 27,580 included a hash value that did not match our reconstructed message hash, suggesting that either a fault occurred during the host's hashing process or the values incorporated into the hash were corrupted in transit. The remaining errors, which included formatting issues or incorrect ASN.1 information, were rare.

4.3 Passive Data Collection and Analysis

In addition to our active SSH scans, we collected data from two network taps at the University of California, San Diego. Both taps

[1] CVE-2022-20866

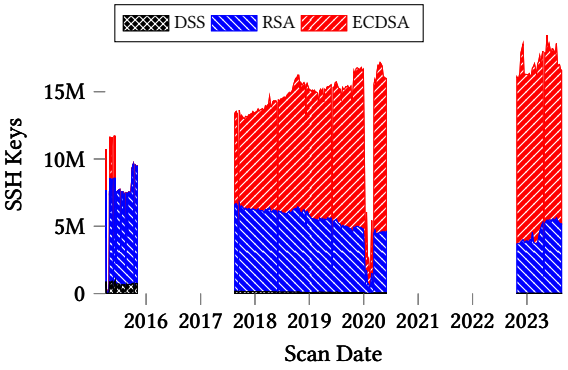


Figure 1: Signature key types over time. We analyzed a heterogeneous dataset of historical and current SSH scans. The prevalence of RSA has decreased among host keys collected over time. We exclude key types, such as Ed25519, that represent less than 1% of each scan.

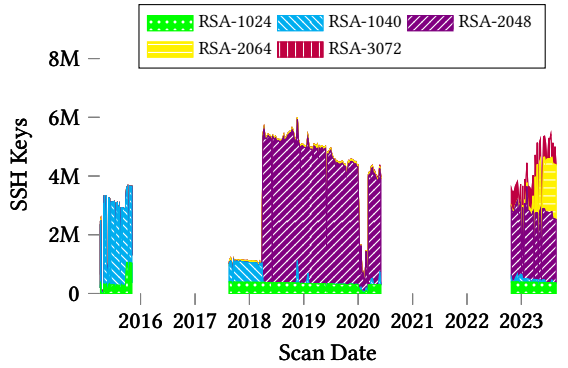


Figure 2: RSA key sizes over time. As RSA has become less common, key lengths have grown. We exclude key sizes that account for fewer than 5% of keys in every scan.

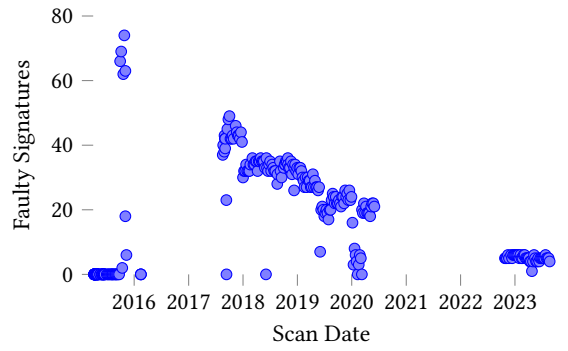


Figure 3: RSA key compromise via signatures over time. The rate of key compromise from vulnerable signatures has decreased over time.

were set up using the same stack. We processed incoming traffic using PF_RING [44] and the Bro [45] Intrusion Detection System [2].

[2]Now known as The Zeek Network Security Monitor: https://zeek.org/. We used an older version, Bro 2.5.3.

Table 5: Common key types in passive data collection.

Signature Type	WiFi
ecdsa-sha2-nistp256	386,259 **(64%)**
ssh-rsa	134,686 **(22%)**
ssh-ed25519	46,336 **(7.7%)**
rsa-sha2-512	34,299 **(5.7%)**
Total	**602,491**

While fully passive attacks are possible for both SSH and IKEv1 in Aggressive mode, Bro does not include a module for analyzing IKEv1 traffic, and the complexity of the numerous IKEv1 configurations made it infeasible to implement our own. For this reason, our analysis of passive data focuses exclusively on SSH connections.

The first tap was on the campus WiFi network and collected data from February 2022 until September 2023. Over this time we saw 2,326,378 connections that Bro identified as SSH traffic. In 602,491 (26%) of these connections we saw the server's signature for the handshake.

Of these connections Bro identified 78% as being outbound connections, 3% as inbound, and the remaining 19% were unidentified, mostly intra-campus traffic.

We checked for valid padding on all of the RSA signatures and did not observe any invalid padding. We are unable to validate the ECDSA signatures we observe in this data because we do not have the hash.

We give the distribution of key types in Table 5. RSA signatures accounted for around 28% of host key signatures. OpenSSH is by far the most common SSH version string we see in our dataset for connections originating on the network. We saw a few thousand connections to hosts with Cisco version strings, although with different versions than the one that generated the faulty signature.

4.4 Active IPsec Data Collection

We carried out internet-wide scans of IKEv1 and IKEv2 hosts to collect certificates and digital signatures. There is no existing official ZGrab2 module for IKE. We implemented a custom module for IKEv1 and IKEv2 to collect data from hosts. The details of collecting this information differ for both versions of the protocol. We focused on authentication modes that would allow us to collect signatures.

4.4.1 IKEv1 protocol. We carried out weekly scans of IKEv1 from November 16, 2022, to August 30, 2023. To find hosts responding to IKEv1, we first did a full IPv4 scan on UDP port 500 using the ZMap scanner with a fixed probe packet offering an IKEv1 security association containing a list of common cipher choices. For each responding host, we used our ZGrab2 module to carry out a partial IKEv1 handshake with that host, using digital signature authentication in aggressive mode. We chose to use aggressive mode because this is the only mode in which a responder authenticates before the initiator.

The initial message in an aggressive mode handshake includes Security Association (SA), Key Exchange (KE), nonce, and Identity (IDii) payloads. For our identity, we used an email address identifying the connection as research from our institution, which recipients could contact to opt out of our scans. An important caveat of

this approach is that IKE responders often use the identity of the initiator (IDii) to lookup IKE policy. We found that many hosts respond with a notification payload indicating `AUTHENTICATION-FAILED` when the identity is not known to the responder. A scan with a more common identity would have likely had a higher response rate, but we wanted to be transparent about our identity.

Our initial message included a Certificate Request (CERTREQ) payload, which is allowed to be sent in any message. We included this because we observed that some responders were configured to only send a Certificate (CERT) payload if the initiator explicitly requests it. For successful connections, the responder responds with SA, KE, nonce, and identity payloads. In addition, successful connections included a Signature (SIG) payload and a Certificate (CERT) payload in response to our request. We terminated the handshake at this point.

As an example of representative response rates, on August 30, 2023, we received a total of 6,151,330 responses from ZMap. When we scanned again with ZGrab2, 6,041,029 hosts did not respond or sent a notification payload indicating an error. See Table 9 for a categorization of all the hosts that responded to the initial ZMap request.

4.4.2 IKEv2 protocol. We carried out weekly scans of IKEv2 from September 15, 2022 to September 1, 2023. To find hosts responding to IKEv2, we first did a full IPv4 scan on UDP port 500 using the ZMap scanner with a fixed probe packet offering an IKEv2 security association containing a list of common cipher suites. For each responding host, we used our ZGrab2 module to carry out a partial IKEv2 handshake with that host, using EAP authentication mode. We chose to scan EAP mode because the responder is authenticated before the initiator using a digital signature. This means our scanner can extract a signature from a host without completing the EAP authentication process.

Our scanner sends initial Security Association (SA), Key Exchange (KE), and Nonce payloads and receives the corresponding values from the responder. In the next exchange, encrypted under session key, our scanner sends sends an Identity (IDi) payload, a CERTREQ payload, and omits the authentication (AUTH) payload to indicate that it wishes to use EAP mode. In a successful response, the responder sends its encrypted Identity, Certificate, Authentication, and EAP payload. We end the connection there.

Our scanner still needs to send IDi in the second round of exchange, and a responder might reject that identity. Interestingly, such rejections are less common than in IKEv1, as can be seen by comparing the percentages for RSA in Tables 9 and 10. One possible reason is that the EAP authentication process will also ask for the identity, and RFC 7296 specifies in section 2.16 that "When the initiator authentication uses EAP, it is possible that the contents of the IDi payload is used only for Authentication, Authorization, and Accounting (AAA) routing purposes and selecting which EAP method to use. This value may be different from the identity authenticated by the EAP method."

As a representative example scan, on September 1, 2023, we received 8,682,630 total responses from our ZMap probes. When we scanned again with ZGrab2, 5,891,901 hosts did not respond or sent a notification payload indicating an error in the first `IKE_SA_INIT` message, and 2,121,067 further hosts did not reply to the second encrypted `IKE_AUTH` message. We received 524,002 RSA signatures, 84,613 ECDSA signatures, and 59,657 PSK-MAC authentication messages in response to our probe. A breakdown of the authentication methods used by IKEv2 hosts that responded to the ZMap probes is in Table 10. Every authentication method except **PSK-MAC** is an asymmetric digital signature scheme.

Fragmentation. IKEv2 allows initiators to specify whether they support fragmentation as specified in RFC 7383. Some form of fragmentation is necessary because many networks have a Maximum Transmissible Unit (MTU) that limits how big a packet it can receive, and our scan requests certificates containing large public keys (especially RSA keys) that often exceeds the size of the MTU. The IP protocol itself supports fragmentation, but it may not always work since many devices filter out IP fragments. Unlike IP fragmentation, IKEv2 fragmentation is performed at the application layer and is specifically designed to avoid needing to use IP fragmentation.

We hypothesized that fragmentation might influence the distribution of key sizes we observed, but the differences across scans were very small.

4.5 IPsec analysis

4.5.1 IKEv1. We collected weekly scans of IKEv1 data from November 16, 2022, to August 30, 2023. IKEv1 gives far fewer signatures compared to IKEv2. Nevertheless, in the IKEv1 scan data for our most recent scan on August 30, 2023 (Table 9), we collected one non-validating signature out of 2,517 total RSA signatures. We collected 83 non-validating signatures across all of our scans. None of these invalid signatures had valid PKCS#1 v1.5 padding, and none of them revealed a private key.

4.5.2 IKEv2. In the weekly IKEv2 scans from September 15, 2022 to September 1, 2023, the number of hosts we observed that responded on port 500 varied from 7,922,385 to 8,682,630 hosts per scan. When we initiated full handshakes with each of these hosts, we received between 471,806 and 530,812 RSA signatures per scan. Among these signatures we collected, we validated each signature and found between 380 and 951 invalid RSA signatures per scan. In total, we collected 65,002 invalid RSA signatures across all of our scans.

For each invalid signature, we attempted to classify the source of failure by checking whether the message recovered from the signature ($s^e \bmod N$) had proper PKCS#1 v1.5 signature padding.

Signatures with incorrect PKCS#1 v1.5 padding. For all but ten of the invalid signatures, the recovered message did not satisfy PKCS#1 v1.5 padding, and appeared to be random. For these signatures, we had the messages since we were a participant in the protocol, and we attempted to factor the RSA modulus using the GCD attack. The attack did not succeed for any of these invalid signatures.

Incorrect RSA moduli. We traced nearly all of these invalid signatures to mismatches between the public key in the host certificate and the private key used to generate the signatures. This type of failure can be detected by taking several invalid signature pairs (m_i, s_i) from a single host and computing $N' = \gcd(s_1^e - m_1, s_2^e - m_2, \dots)$. For failures of this type, N' is typically the real value of the public modulus N with high probability. For other types of failures, typically $N' = 1$. See Appendix A.1 for a more detailed justification.

For 576 out of the 579 invalid signatures collected on January 6, 2023, we recovered proper moduli which make the signatures valid. About half of the certificate subjects for the affected devices identified a Watchguard VPN product.

Signatures with correct or nearly correct PKCS#1 v1.5 padding. We observed a small number of invalid signatures where the recovered message had correct PKCS#1 v1.5 or PKCS#1 v1.5-like padding. Our September 15, 2022 IKEv2 scan results contained four signatures that did not validate because s^e mod N contains just the PKCS#1 v1.5 padding of the SHA-1 hash *without* the ASN.1 OID preceding the hash. That is, s^e = `00 || 01 || FF ... FF || 00 || SHA-1(m)`. The hash is otherwise a valid SHA-1 hash of the message to be signed. This is consistent with the IKEv1 signature format where there is no OID, so this indicates that an implementation may have mixed up the formats between the two IKE versions.

There are also six instances of s^e mod N being in the correct format (PKCS#1 v1.5 padding followed by ASN.1 OID and then the hash), but the hash does not match that of the signed message. This indicates that either the hash or handshake messages were corrupted either in transit or during computation.

PSK vulnerabilities. Several IKEv2 hosts responded with a PSK authentication (AUTH) message containing a MAC derived from a pre-shared key, see Section 2.5. For example, in our September 7, 2022 scan, out of 7,925,522 total responses to our ZMap probe, 55,936 returned this type of response. If the pre-shared key is set to a poorly chosen password, knowledge of this AUTH payload allows a dictionary attack to determine which key generated the MAC. We checked these unwittingly collected MACs against the rockyou wordlist [37], and found that 8,868 (16%) used a pre-shared key matching a password from the rockyou list.

5 DISCUSSION

5.1 Countermeasures

The countermeasure to the attacks we describe in this paper is well known: implementations should validate signatures before sending them. OpenSSH, the most common SSH implementation we observed in this data, implements this countermeasure because it uses OpenSSL to generate signatures, and OpenSSL has included countermeasures against RSA fault attacks since 2001.

5.2 OpenSSH deprecates `ssh-rsa`

OpenSSH deprecated the `ssh-rsa` signature scheme in version 8.8, released in September 2021. This deprecation does not remove the use of RSA entirely, only the `ssh-rsa` signature type that uses the SHA-1 hash function. Signature types of `rsa-sha2-256` and `rsa-sha2-512` using SHA-2 are still enabled in OpenSSH. However, our passive network data shows that `ssh-rsa` remains more common than the `rsa-sha2` alternatives, and both types represent almost 20% of connections we observe.

5.3 Lessons for protocol design

These attacks provide a concrete illustration of the value of several design principles in cryptography: encrypting protocol handshakes as soon as a session key is negotiated to protect metadata, binding authentication to a session, and separating authentication from encryption keys. TLS 1.3 has taken these steps, as have versions of IPsec. Although an active attacker can still make a connection to the server in hopes of triggering a fault, the attack is no longer passive, and a server would have records of, for example, a large number of connections.

5.4 Limitations

Our visibility into the IPsec protocol is particularly limited, given the constraints on our data collection: the number of signatures we were able to collect is a small fraction of the dataset we were able to collect for SSH. Given the rarity of vulnerable signature faults, we are not able to conclude much about IPsec implementations from our data, and believe this question deserves further study.

ACKNOWLEDGMENTS

We are grateful to Zakir Durumeric for providing us with historical scan data from the University of Michigan and Censys. This work was supported by the National Science Foundation under grants no. 2048563 and 1913210.

REFERENCES

[1] N. Asokan, Valtteri Niemi, and Kaisa Nyberg. 2002. Man-in-the-Middle in Tunnelled Authentication Protocols. Cryptology ePrint Archive, Report 2002/163. https://eprint.iacr.org/2002/163.

[2] Christian Aumüller, Peter Bier, Wieland Fischer, Peter Hofreiter, and Jean-Pierre Seifert. 2003. Fault Attacks on RSA with CRT: Concrete Results and Practical Countermeasures. In *CHES 2002 (LNCS, Vol. 2523)*, Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar (Eds.). Springer, Heidelberg, 260–275. https://doi.org/10.1007/3-540-36400-5_20

[3] Guillaume Barbu, Alberto Battistello, Guillaume Dabosville, Christophe Giraud, Guénaël Renault, Soline Renner, and Rina Zeitoun. 2013. Combined Attack on CRT-RSA - Why Public Verification Must Not Be Public?. In *PKC 2013 (LNCS, Vol. 7778)*, Kaoru Kurosawa and Goichiro Hanaoka (Eds.). Springer, Heidelberg, 198–215. https://doi.org/10.1007/978-3-642-36362-7_13

[4] Gilles Barthe, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Mehdi Tibouchi, and Jean-Christophe Zapalowicz. 2014. Making RSA-PSS Provably Secure against Non-random Faults. In *CHES 2014 (LNCS, Vol. 8731)*, Lejla Batina and Matthew Robshaw (Eds.). Springer, Heidelberg, 206–222. https://doi.org/10.1007/978-3-662-44709-3_12

[5] Mihir Bellare and Phillip Rogaway. 1996. The Exact Security of Digital Signatures: How to Sign with RSA and Rabin. In *EUROCRYPT'96 (LNCS, Vol. 1070)*, Ueli M. Maurer (Ed.). Springer, Heidelberg, 399–416. https://doi.org/10.1007/3-540-68339-9_34

[6] Jingguo Bi, Jean-Sébastien Coron, Jean-Charles Faugère, Phong Q. Nguyen, Guénaël Renault, and Rina Zeitoun. 2014. Rounding and Chaining LLL: Finding Faster Small Roots of Univariate Polynomial Congruences. In *PKC 2014 (LNCS, Vol. 8383)*, Hugo Krawczyk (Ed.). Springer, Heidelberg, 185–202. https://doi.org/10.1007/978-3-642-54631-0_11

[7] D. Bider. 2018. RFC 8332: Use of RSA Keys with SHA-256 and SHA-512 in the Secure Shell (SSH) Protocol.

[8] Johannes Blömer, Martin Otto, and Jean-Pierre Seifert. 2003. A New CRT-RSA Algorithm Secure Against Bellcore Attacks. In *ACM CCS 2003*, Sushil Jajodia, Vijayalakshmi Atluri, and Trent Jaeger (Eds.). ACM Press, 311–320. https://doi.org/10.1145/948109.948151

[9] Hanno Böck, Juraj Somorovsky, and Craig Young. 2018. Return Of Bleichenbacher's Oracle Threat (ROBOT). In *27th USENIX Security Symposium (USENIX Security 18)*. USENIX Association, Baltimore, MD, 817–849. https://www.usenix.org/conference/usenixsecurity18/presentation/bock

[10] Dan Boneh et al. 1999. Twenty years of attacks on the RSA cryptosystem. *Notices of the AMS* 46, 2 (1999), 203–213.

[11] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. 2001. On the Importance of Eliminating Errors in Cryptographic Computations. *Journal of Cryptology* 14, 2 (March 2001), 101–119. https://doi.org/10.1007/s001450010016

[12] Joachim Breitner and Nadia Heninger. 2019. Biased Nonce Sense: Lattice Attacks Against Weak ECDSA Signatures in Cryptocurrencies. In *FC 2019 (LNCS, Vol. 11598)*, Ian Goldberg and Tyler Moore (Eds.). Springer, Heidelberg, 3–20. https://doi.org/10.1007/978-3-030-32101-7_1

[13] Eric Brier, David Naccache, Phong Q. Nguyen, and Mehdi Tibouchi. 2011. Modulus Fault Attacks against RSA-CRT Signatures. In *CHES 2011 (LNCS, Vol. 6917)*, Bart

Preneel and Tsuyoshi Takagi (Eds.). Springer, Heidelberg, 192–206. https://doi.org/10.1007/978-3-642-23951-9_13
[14] Don Coppersmith. 1996. Finding a Small Root of a Bivariate Integer Equation; Factoring with High Bits Known. In *EUROCRYPT'96 (LNCS, Vol. 1070)*, Ueli M. Maurer (Ed.). Springer, Heidelberg, 178–189. https://doi.org/10.1007/3-540-68339-9_16
[15] Don Coppersmith. 1997. Small Solutions to Polynomial Equations, and Low Exponent RSA Vulnerabilities. *Journal of Cryptology* 10, 4 (Sept. 1997), 233–260. https://doi.org/10.1007/s001459900030
[16] Jean-Sébastien Coron, Antoine Joux, Ilya Kizhvatov, David Naccache, and Pascal Paillier. 2009. Fault Attacks on RSA Signatures with Partially Unknown Messages. In *CHES 2009 (LNCS, Vol. 5747)*, Christophe Clavier and Kris Gaj (Eds.). Springer, Heidelberg, 444–456. https://doi.org/10.1007/978-3-642-04138-9_31
[17] Jean-Sébastien Coron, Antoine Joux, Ilya Kizhvatov, David Naccache, and Pascal Paillier. 2009. Fault Attacks on RSA Signatures with Partially Unknown Messages. Cryptology ePrint Archive, Report 2009/309. https://eprint.iacr.org/2009/309.
[18] T. Dierks and E. Rescorla. 2008. RFC 5246: The Transport Layer Security (TLS) Protocol Version 1.2.
[19] Kristen Dorey, Nicholas Chang-Fong, and Aleksander Essex. 2017. Indiscreet Logs: Diffie-Hellman Backdoors in TLS. In *NDSS 2017*. The Internet Society.
[20] Zakir Durumeric, David Adrian, Ariana Mirian, Michael Bailey, and J. Alex Halderman. 2015. A Search Engine Backed by Internet-Wide Scanning. In *ACM CCS 2015*, Indrajit Ray, Ninghui Li, and Christopher Kruegel (Eds.). ACM Press, 542–553. https://doi.org/10.1145/2810103.2813703
[21] Zakir Durumeric, Eric Wustrow, and J. Alex Halderman. 2013. ZMap: Fast Internet-wide Scanning and Its Security Applications. In *USENIX Security 2013*, Samuel T. King (Ed.). USENIX Association, 605–620.
[22] Pierre-Alain Fouque, Nicolas Guillermin, Delphine Leresteux, Mehdi Tibouchi, and Jean-Christophe Zapalowicz. 2013. Attacking RSA-CRT signatures with faults on montgomery multiplication. *Journal of Cryptographic Engineering* 3, 1 (April 2013), 59–72. https://doi.org/10.1007/s13389-013-0050-x
[23] Lidong Han, Wei Wei, and Mingjie Liu. 2013. On the Multiple Fault Attacks on RSA Signatures with LSBs of Messages Unknown. In *Information Security and Cryptology*, Mirosław Kutyłowski and Moti Yung (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 1–9.
[24] Dan Harkins and Dave Carrel. 1998. The Internet Key Exchange (IKE). IETF RFC 2409 (Proposed Standard).
[25] B. Harris. 2006. RFC 4432: RSA Key Exchange for the Secure Shell (SSH) Transport Layer Protocol.
[26] Marcella Hastings, Joshua Fried, and Nadia Heninger. 2016. Weak Keys Remain Widespread in Network Devices. In *Proceedings of the 2016 Internet Measurement Conference (IMC '16)*. Association for Computing Machinery, New York, NY, USA, 49–63. https://doi.org/10.1145/2987443.2987486
[27] Nadia Heninger, Zakir Durumeric, Eric Wustrow, and J. Alex Halderman. 2012. Mining Your Ps and Qs: Detection of Widespread Weak Keys in Network Devices. In *USENIX Security 2012*, Tadayoshi Kohno (Ed.). USENIX Association, 205–220.
[28] Mathias Herrmann and Alexander May. 2008. Solving Linear Equations Modulo Divisors: On Factoring Given Any Bits. In *ASIACRYPT 2008 (LNCS, Vol. 5350)*, Josef Pieprzyk (Ed.). Springer, Heidelberg, 406–424. https://doi.org/10.1007/978-3-540-89255-7_25
[29] Nick Howgrave-Graham. 2001. Approximate Integer Common Divisors. In *Cryptography and Lattices*, Joseph H. Silverman (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 51–66.
[30] Tibor Jager, Saqib A. Kakvi, and Alexander May. 2018. On the Security of the PKCS#1 v1.5 Signature Scheme. In *ACM CCS 2018*, David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang (Eds.). ACM Press, 1195–1208. https://doi.org/10.1145/3243734.3243798
[31] Burt Kaliski. 1998. PKCS# 1: RSA encryption version 1.5.
[32] C. Kaufman, P. Hoffman, Y. Nir, P. Eronen, and T. Kivinen. 2014. RFC 7296: Internet Key Exchange Protocol Version 2 (IKEv2).
[33] Chong Hee Kim and Jean-Jacques Quisquater. 2007. Fault attacks for CRT based RSA: New attacks, new results, and new countermeasures. In *IFIP International Workshop on Information Security Theory and Practices*. Springer, 215–228.
[34] Paul Kirchner, Thomas Espitau, and Pierre-Alain Fouque. 2021. Towards Faster Polynomial-Time Lattice Reduction. In *CRYPTO 2021, Part II (LNCS, Vol. 12826)*, Tal Malkin and Chris Peikert (Eds.). Springer, Heidelberg, Virtual Event, 760–790. https://doi.org/10.1007/978-3-030-84245-1_26
[35] Arjen K Lenstra. 1996. *Memo on RSA signature generation in the presence of faults*. Technical Report. EPFL. https://infoscience.epfl.ch/record/164524.
[36] A. K. Lenstra, H. W. Lenstra, and L. Lovász. 1982. Factoring polynomials with rational coefficients. *Math. Ann.* 261, 4 (Dec 1982), 515–534. https://doi.org/10.1007/BF01457454
[37] Kali Linux. 2022. Wordlists: Tool Documentation. https://www.kali.org/tools/wordlists/.
[38] Moxie Marlinspike, David Hulton, and Marsh Ray. 2012. Defeating PPTP VPNs and WPA2 Enterprise with MS-CHAPv2. In *DEFCON 20*. DEFCON. https://media.defcon.org/DEF%20CON%2020/DEF%20CON%2020%20video%20and%20slides/DEF%20CON%2020%20-%20Marlinspike%20Hulton%20and%20Ray%20-%20Defeating%20PPTP%20VPNs%20and%20WPA2%20Enterprise%20with%20MS-CHAPv2%20-%20Video%20and%20Slides.mp4
[39] D. Maughan, M. Schertler, M. Schneider, and J. Turner. 1998. RFC 2408: Internet Security Association and Key Management Protocol (ISAKMP).
[40] Alexander May. 2009. Using LLL-reduction for solving RSA and factorization problems. In *The LLL algorithm*. Springer, 315–348.
[41] Robert Merget, Juraj Somorovsky, Nimrod Aviram, Craig Young, Janis Fliegenschmidt, Jörg Schwenk, and Yuval Shavitt. 2019. Scalable Scanning and Automatic Classification of TLS Padding Oracle Vulnerabilities. In *USENIX Security 2019*, Nadia Heninger and Patrick Traynor (Eds.). USENIX Association, 1029–1046.
[42] Damien Miller. 2022. SSH agent restriction. https://www.openssh.com/agent-restrict.html.
[43] S. Moonesamy. 2015. RFC 7479: Using Ed25519 in SSHFP Resource Records.
[44] ntop. 2023. PF_RING: High-speed packet capture, filtering and analysis. https://www.ntop.org/products/packet-capture/pf_ring/.
[45] Vern Paxson. 1999. Bro: A System for Detecting Network Intruders in Real-Time. *Comput. Netw.* 31, 23–24 (December 1999), 2435–2463. https://doi.org/10.1016/S1389-1286(99)00112-7
[46] Andrea Pellegrini, Valeria Bertacco, and Todd Austin. 2010. Fault-based attack of RSA authentication. In *2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010)*. IEEE, 855–860.
[47] E. Rescorla. 2018. RFC 8446: The Transport Layer Security (TLS) Protocol Version 1.3.
[48] Keegan Ryan and Nadia Heninger. 2023. Fast Practical Lattice Reduction Through Iterated Compression. In *Advances in Cryptology – CRYPTO 2023*, Helena Handschuh and Anna Lysyanskaya (Eds.). Springer Nature Switzerland, Cham, 3–36.
[49] D. Stebila and J. Green. 2009. RFC 5656: Elliptic Curve Algorithm Integration in the Secure Shell Transport Layer.
[50] George Arnold Sullivan, Jackson Sippe, Nadia Heninger, and Eric Wustrow. 2022. Open to a fault: On the passive compromise of TLS keys via transient errors. In *31st USENIX Security Symposium (USENIX Security 22)*. USENIX Association, Boston, MA, 233–250. https://www.usenix.org/conference/usenixsecurity22/presentation/sullivan
[51] Luke Valenta, David Adrian, Antonio Sanso, Shaanan Cohney, Joshua Fried, Marcella Hastings, J. Alex Halderman, and Nadia Heninger. 2017. Measuring small subgroup attacks against Diffie-Hellman. In *NDSS 2017*. The Internet Society.
[52] Luke Valenta, Nick Sullivan, Antonio Sanso, and Nadia Heninger. 2018. In Search of CurveSwap: Measuring Elliptic Curve Implementations in the Wild. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*. 384–398. https://doi.org/10.1109/EuroSP.2018.00034
[53] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. 2010. Fully Homomorphic Encryption over the Integers. In *EUROCRYPT 2010 (LNCS, Vol. 6110)*, Henri Gilbert (Ed.). Springer, Heidelberg, 24–43. https://doi.org/10.1007/978-3-642-13190-5_2
[54] Florian Weimer. 2015. *Factoring RSA Keys With TLS Perfect Forward Secrecy*. Technical Report. Red Hat. https://www.redhat.com/en/blog/factoring-rsa-keys-tls-perfect-forward-secrecy.
[55] Zane Weissman, Thore Tiemann, Daniel Moghimi, Evan Custodio, Thomas Eisenbarth, and Berk Sunar. 2020. JackHammer: Efficient Rowhammer on Heterogeneous FPGA-CPU Platforms. *IACR TCHES* 2020, 3 (2020), 169–195. https://doi.org/10.13154/tches.v2020.i3.169-195 https://tches.iacr.org/index.php/TCHES/article/view/8587.
[56] T. Ylonen and C. Lonvick. 2006. RFC 4252: The Secure Shell (SSH) Authentication Protocol.
[57] T. Ylonen and C. Lonvick. 2006. RFC 4253: The Secure Shell (SSH) Transport Layer Protocol.

A APPENDIX

A.1 Incorrect modulus recovery from signatures

We conjectured that some invalid signatures were due to a host not presenting the correct RSA public key (N, e). To test this, we assumed $e = 65537$, since this was the most common exponent in our data, and used ten pairs of invalid signatures and padded messages $(s_1, m_1), (s_2, m_2), \ldots$ from the same host collected in follow-up scans. We then computed $N' = \gcd(s_1^e - m_1, s_2^e - m_2, \ldots)$.

Since $s_i^e = m_i \bmod N$, then N divides $s_i^e - m_i$ over the integers. This implies N divides N'. In practice, the values of $s_i^e - m_i$ are unlikely to share other factors, and $N = N'$ with high probability. If the signatures were invalid for a reason other than the host presenting an incorrect modulus, then the values of $s_i^e - m_i$ are unlikely to all share a common factor, and in this case $N' = 1$.

A.2 Additional Tables

Table 6: Key sizes in passive SSH data collection.

Signature Type	Key Size	WiFi
ecdsa-sha2-nistp256	-	**386,259 (64%)**
ssh-rsa	**all**	**134,686 (22%)**
	3072	*61,626*
	2048	*60,622*
	1024	*12,018*
	1040	*360*
	4096	*57*
	1536	*2*
	8192	*1*
ssh-ed25519	-	**46,336 (7.7%)**
rsa-sha2-512	**all**	**34,299 (5.7%)**
	2048	*33,406*
	1024	*852*
	3072	*24*
	4096	*17*
ssh-dss	-	**271 (0.0%)**
rsa-sha2-256	**all**	**601 (0.0%)**
	2048	*408*
	3072	*190*
	4096	*3*
ecdsa-sha2-nistp521	-	**36 (0.0%)**
ecdsa-sha2-nistp384	-	**3 (0.0%)**
Total		**602,491**

Table 7: Most Common Passive WiFi Data SSH Clients.

SSH Version String	Count
SSH-2.0-OpenSSH_7.6p1 Ubuntu-4ubuntu0.3	191,921
SSH-2.0-OpenSSH_7.2p2 Ubuntu-4ubuntu2.8	150,869
SSH-2.0-SecureBlackbox	57,069
SSH-2.0-OpenSSH_8.6	27,692
SSH-2.0-OpenSSH_8.1	22,248
SSH-2.0-Go	18,436
SSH-2.0-PuTTY_Release_0.70.2_Sourcetree	18,318
SSH-2.0-SecureBlackbox.9	12,823
SSH-2.0-libssh2_1.9.0_DEV	7,858
SSH-2.0-OpenSSH_9.0p1 Debian-1	7,786

Table 8: Most Common Passive WiFi Data SSH Servers.

SSH Version String	Count
SSH-2.0-OpenSSH_7.2p2 Ubuntu-4ubuntu2.8	345,858
SSH-2.0-OpenSSH_7.4	50,586
SSH-2.0-OpenSSH_8.2p1	48,592
SSH-2.0-OpenSSH_4.3	11,928
SSH-2.0-Go	8,983
SSH-2.0-OpenSSH_7.2p2 Ubuntu-4ubuntu2.10	8,880
SSH-2.0-OpenSSH_7.5p1b-GSI NMOD_3.19...	8,263
SSH-2.0-Cisco-1.25	6,372
SSH-2.0-OpenSSH_6.6.1	5,632
SSH-2.0-OpenSSH_7.5 PKIX[10.1]	5,522

Table 9: IKEv1 scanning outcomes for August 30, 2023. IKEv1 supports RSA or DSS signatures, but we only saw RSA.

Handshake outcome	Key Size	Connections
Rejected handshake	-	**6,041,029 (98.21%)**
Missing signature	-	**107,169 (1.74%)**
RSA signature	**all**	**2,517 (0.04%)**
	2048	2,145
	1024	344
	4096	25
	512	2
	3072	1
Missing public key	-	**615 (0.01%)**
Total		**6,151,330**

Table 10: IKEv2 scanning outcomes for September 1, 2023.

Handshake outcome	Key Size	Connections
Rejected handshake	-	**5,891,901 (67.86%)**
Missing `IKE_AUTH`	-	**2,121,067 (24.43%)**
RSA signature	**all**	**524,002 (6.04%)**
ECDSA signature	**all**	**84,613 (0.97%)**
PSK-MAC	-	**59,657 (0.69%)**
Missing signature	-	**993 (0.01%)**
Missing public key	-	**329 (0.00%)**
DSS signature	-	**68 (0.00%)**
Total		**8,682,630**

Table 11: IKEv2 key lengths in the September 1, 2023 scan. The "other" category includes keys of bit length 2192, 2432, 3584, 4104, 6144, 7680, and 16384.

Key Type	Key Size	Connections
RSA	**all**	**524,002 (6.04%)**
	512	121
	1024	17,215
	2048	420,270
	3072	630
	4096	85,681
	8192	65
	other	20
ECDSA	**all**	**84,613 (0.97%)**
	256	84,386
	384	189
	521	38

Received 19 January 2023