usenix
THE ADVANCED
COMPUTING SYSTEMS
ASSOCIATION

The Dangers of Key Reuse: Practical Attacks on IPsec IKE

Dennis Felsch, Martin Grothe, and Jörg Schwenk, *Ruhr-University Bochum;*
Adam Czubak and Marcin Szymanek, *University of Opole*

https://www.usenix.org/conference/usenixsecurity18/presentation/felsch

This paper is included in the Proceedings of the
27th USENIX Security Symposium.

August 15–17, 2018 • Baltimore, MD, USA

ISBN 978-1-939133-04-5

The Dangers of Key Reuse: Practical Attacks on IPsec IKE

Dennis Felsch
Ruhr-University Bochum
dennis.felsch@rub.de

Martin Grothe
Ruhr-University Bochum
martin.grothe@rub.de

Jörg Schwenk
Ruhr-University Bochum
joerg.schwenk@rub.de

Adam Czubak
University of Opole
aczubak@uni.opole.pl

Marcin Szymanek
University of Opole
mszymanek@uni.opole.pl

Abstract

IPsec enables cryptographic protection of IP packets. It is commonly used to build VPNs (Virtual Private Networks). For key establishment, the IKE (Internet Key Exchange) protocol is used. IKE exists in two versions, each with different modes, different phases, several authentication methods, and configuration options.

In this paper, we show that reusing a key pair across different versions and modes of IKE can lead to cross-protocol authentication bypasses, enabling the impersonation of a victim host or network by attackers. We exploit a Bleichenbacher oracle in an IKEv1 mode, where RSA encrypted nonces are used for authentication. Using this exploit, we break these RSA *encryption* based modes, and in addition break RSA *signature* based authentication in both IKEv1 and IKEv2. Additionally, we describe an offline dictionary attack against the PSK (Pre-Shared Key) based IKE modes, thus covering all available authentication mechanisms of IKE.

We found Bleichenbacher oracles in the IKEv1 implementations of Cisco (CVE-2018-0131), Huawei (CVE-2017-17305), Clavister (CVE-2018-8753), and ZyXEL (CVE-2018-9129). All vendors published fixes or removed the particular authentication method from their devices' firmwares in response to our reports.

1 Introduction

VPNs (Virtual Private Networks) allow employees to securely access a corporate network while they are outside the office. They also allow companies to connect their local networks over the public Internet. Examples for large industrial VPNs are the ANX (Automotive Network Exchange), ENX (European Network Exchange), and JNX (Japanese Network Exchange) associations, which connect vehicle manufacturers with their suppliers [1–3]. In 4G/LTE (Long Term Evolution) networks, wireless carriers use VPNs to secure the backhaul links between base stations and the core network [4, pp. 66–67]. Other applications of VPNs involve circumventing geo-restrictions and censorship.

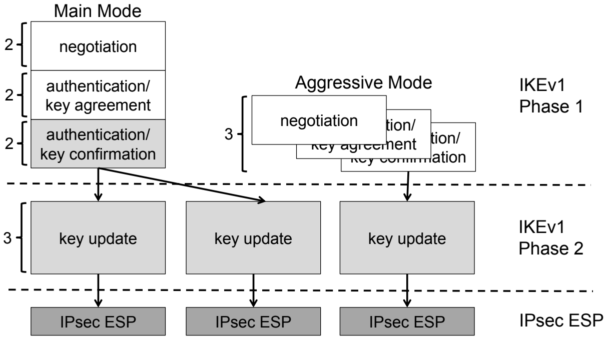

Figure 1: The relationship between IKEv1 Phase 1, Phase 2, and IPsec ESP. Multiple simultaneous Phase 2 connections can be established from a single Phase 1 connection. Grey parts are encrypted, either with IKE derived keys (light grey) or with IPsec keys (dark grey). The numbers at the curly brackets denote the number of messages to be exchanged in the protocol.

IPsec (Internet Protocol Security) is a protocol stack that protects network packets at the IP layer. In contrast to other widespread cryptographic protocols like TLS (Transport Layer Security) or SSH (Secure Shell), which operate at the application layer, IPsec allows to protect *every* IP based communication. When transmitting payload data, IPsec uses two different data formats to protect IP packets: AH (Authentication Header) for integrity-only setups and ESP (Encapsulating Security Payload) for confidentiality with optional integrity.

IKE. To establish a shared secret for an IPsec connection, the IKE protocol has to be executed. There are two different versions of IKE named IKEv1 (1998) and IKEv2 (2005). Although IKEv2 officially obsoletes the

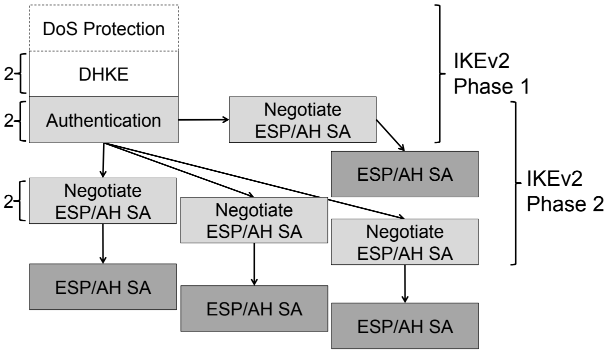

Figure 2: The relationship between IKEv2 Phase 1, Phase 2, and IPsec ESP. Multiple simultaneous Phase 2 connections can be established from a single Phase 1 connection. Furthermore, Phase 1 and Phase 2 are partially interleaved. Grey parts are encrypted, either with IKE derived keys (light grey) or with IPsec keys (dark grey). The numbers at the curly brackets to the left denote the number of messages to be exchanged in the protocol.

previous version, both are still available in all implementations and both can be configured for actual use in all major operating systems and network devices.

IKE consists of two phases, where Phase 1 is used to establish initial authenticated keying material between two peers. Phase 2 is used to negotiate further derived keys for many different IP based connections between the two.

IKE is one of the most complex protocols in use, and the dependencies between Phase 1 and Phase 2 make it hard to analyze. Figures 1 and 2 illustrate this complexity: In IKEv1, both phases are clearly separated, but there are two different modes for Phase 1. In IKEv2, Phase 1 has been simplified, but now Phase 1 interleaves with the first execution of the Phase 2 protocol.

Authentication. In IKEv1, four authentication methods are available for Phase 1 (cf. subsection 2.2): Two RSA encryption based methods, one signature based method, and a PSK (Pre-Shared Key) based method. All Phase 1 modes/methods contain a DHKE (Diffie-Hellman Key Exchange), which guarantees PFS (Perfect Forward Secrecy) for every connection. IKEv2 Phase 1 omits both encryption-based authentication methods, so only signature and PSK based authentication remain.

Attacks. Our attacks only target Phase 1 in IKEv1 and IKEv2, where we impersonate an IKE device. Once attackers succeed with this attack on Phase 1, they share a set of (falsely) authenticated symmetric keys with the victim device, and can successfully complete Phase 2 – this holds for both IKEv1 and IKEv2. The attacks are based on Bleichenbacher oracles discovered in implementations of the two RSA encryption based IKEv1 variants (cf. sections 5–7). These Bleichenbacher oracles can very efficiently be used to decrypt nonces, which breaks these two variants (subsection 4.2). The oracles can also be used to forge digital signatures, which breaks the signature based IKEv1 and IKEv2 variants (subsection 4.4).

We additionally show that both PSK based modes can be broken with an offline dictionary attack if the PSK has low entropy (section 9). We thus provide attacks against *all* authentication modes in both IKEv1 and IKEv2 under reasonable assumptions.

Contribution. In this paper, we make the following contributions:

- We identify and describe Bleichenbacher oracles in the IKEv1 implementations of four large network equipment manufacturers, Cisco, Huawei, Clavister, and ZyXEL.
- We show that the strength of these oracles is sufficient to break *all* handshake variants in IKEv1 and IKEv2 (except those based on PSKs) when given access to powerful network equipment.
- We demonstrate that key reuse across protocols as implemented in certain network equipment carries high security risks.
- We complete the evaluation of all variants of IKEv1 and IKEv2 by showing that *all* PSK based variants are vulnerable to offline dictionary attacks if low entropy PSKs are used. Such attacks were previously only documented for one out of the three PSK-based variants of IKE.

Responsible Disclosure. We reported our findings to Cisco, Huawei, Clavister, and ZyXEL. Cisco published fixes with IOS XE versions 16.3.6, 16.6.3, and 16.7.1. They further informed us that the vulnerable authentication method would be removed with the next major release. Huawei published firmware version V300R001C10SPH702 for the Secospace USG2000 series that removes the Bleichenbacher oracle and fixes crash bugs we identified on our test device. Customers who use other affected Huawei devices will be contacted directly by their support team as part of a need-to-know strategy. Clavister removed the vulnerable authentication method with cOS version 12.00.09. ZyXEL responded that our ZyWALL USG 100 test device is from a legacy model series that is end-of-support. Therefore, these devices will not receive a fix. For the successor models, the patched firmware version ZLD 4.32 is available.

2 IKE (Internet Key Exchange)

IKE is a family of AKE (Authenticated Key Exchange) protocols. It is responsible for negotiating multiple sets of cryptographic algorithms and keys, called SAs (Security Associations) in IPsec terminology. Each SA can either be used to protect the integrity of IP packets with the data format AH (Authentication Header) or to protect confidentiality with optional integrity using the data format ESP (Encapsulating Security Payload). IKE messages are exchanged over UDP (User Datagram Protocol) and their destination port is 500.

IKE is standardized in two major versions: Version 1, described in RFC 2409 [16] and accompanying documents, was published in 1998. It has been declared obsolete by the IETF (Internet Engineering Task Force), but it is nevertheless included in all implementations and still widely used. Version 2, first published in RFC 4306 in 2005 [22] was designed as a low-latency alternative to Version 1, and therefore has a fundamentally different design. It is subject of ongoing standardization, but only minor clarifications are incorporated in the most recent RFCs. IKEv1 uses a data format called ISAKMP (Internet Security Association and Key Management Protocol), which has later been integrated with IKEv2.

2.1 IKEv1 Phases

IKEv1 consists of two phases (cf. Figure 1). In Phase 1, a SA is established for IKEv1 itself, such that the subsequent Phase 2 messages can be encrypted. Additionally, a shared symmetric key is established as basis of authentication in Phase 2. In Phase 2, several SAs for IPsec AH and ESP are negotiated.

IKEv1 Phase 1. For Phase 1 of the protocol, two modes – main mode and aggressive mode – and four authentication methods are available. A main mode handshake consists of exactly six messages; an aggressive mode handshake compresses the protocol flow into only three messages. We do not cover the aggressive mode explicitly in this paper. However, all results described in this paper hold for the aggressive mode as well. Throughout the rest of this paper, we assume readers familiar with the TLS protocol, as we will sometimes compare IKE with TLS.

Figure 3 gives a simplified overview of the IKE protocol structure of Phase 1. Since IKE uses UDP, the protocol itself has to keep track of the handshake session. IKE uses random values called *cookies* (and denoted by c_I and c_R) for this purpose; these cookies are present in each IKE header.

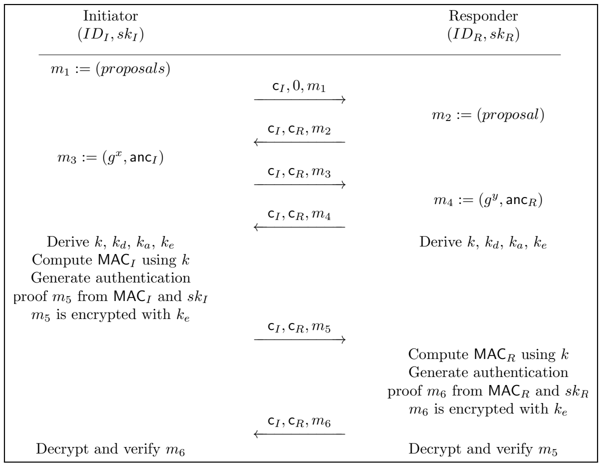
Initiator
(ID_I, sk_I)
Responder
(ID_R, sk_R)
$m_1 := (proposals)$
$\mathsf{c}_I, 0, m_1$
$m_2 := (proposal)$
$\mathsf{c}_I, \mathsf{c}_R, m_2$
$m_3 := (g^x, \mathsf{anc}_I)$
$\mathsf{c}_I, \mathsf{c}_R, m_3$
$m_4 := (g^y, \mathsf{anc}_R)$
$\mathsf{c}_I, \mathsf{c}_R, m_4$
Derive k, k_d, k_a, k_e
Compute MAC_I using k
Generate authentication proof m_5 from MAC_I and sk_I
m_5 is encrypted with k_e
Derive k, k_d, k_a, k_e
$\mathsf{c}_I, \mathsf{c}_R, m_5$
Compute MAC_R using k
Generate authentication proof m_6 from MAC_R and sk_R
m_6 is encrypted with k_e
$\mathsf{c}_I, \mathsf{c}_R, m_6$
Decrypt and verify m_6
Decrypt and verify m_5

Figure 3: Generic structure of IKEv1 Phase 1 in main mode.

The first two messages (m_1 and m_2) are used to negotiate on a *proposal* – a combination of different cryptographic algorithms, comparable to TLS ciphersuites. In messages m_3 and m_4 a DHKE is performed, together with the exchange of additional parameters called *ancillary data* (anc), depending on the chosen authentication method.

Based on these messages and the shared DH secret, four symmetric keys (k, k_d, k_a, k_e)[1] are derived by both parties (cf. Table 1). The formula to derive the intermediate key k varies between the different authentication methods, which are explained in more detail in the following sections. From this intermediate key, the other three keys are derived as the result of a pseudorandom function. Inputs to this function are k, the most recent generated key, the shared DH secret g^{xy}, the cookies $(\mathsf{c}_I, \mathsf{c}_R)$, and an index.

The last two messages (m_5 and m_6) are used for key confirmation. For this, two MAC values[2] are generated using k. These MACs are either exchanged or digitally signed. In main mode, messages m_5 and m_6 are encrypted under key k_e.

	Signature	PKE & RPKE	PSK
k	$\mathsf{PRF}_{n_I,n_R}(g^{xy})$	$\mathsf{PRF}_{\mathsf{h}(n_I,n_R)}(\mathsf{c}_I,\mathsf{c}_R)$	$\mathsf{PRF}_{PSK}(n_I,n_R)$
k_d	$\mathsf{PRF}_k(g^{xy},\mathsf{c}_I,\mathsf{c}_R,0)$		
k_a	$\mathsf{PRF}_k(k_d,g^{xy},\mathsf{c}_I,\mathsf{c}_R,1)$		
k_e	$\mathsf{PRF}_k(k_a,g^{xy},\mathsf{c}_I,\mathsf{c}_R,2)$		

Table 1: The key derivation in the four different authentication methods.

IKEv1 Phase 2. Phase 2 is also called *quick mode*. In essence, quick mode is a three-message PSK based authenticated key agreement protocol. Its security is based on $psk = (k_a, k_d)$ from Phase 1 while key k_e is used to encrypt all messages. For each of the several executions of Phase 2, fresh nonces are exchanged. If PFS is desired, a DHKE can additionally be performed.

2.2 IKEv1 Authentication Methods

In Phase 1 of IKEv1, four different modes of authentication are available: (a) Digital signatures, (b) PKE (Public Key Encryption), (c) RPKE (Revised Public Key Encryption), and (d) PSKs (Pre-Shared Keys). While the message exchange patterns in Phase 1 are fixed to main or aggressive mode, the two communicating entities may freely negotiate any of these four authentication modes.

Signature Based Authentication. This authentication mode assumes that each party owns an asymmetric key pair with valid certificates. After choosing this authentication mode, nonces n_I and n_R are exchanged as ancillary information with the third and fourth message. These nonces are then used as key input to the PRF function, which is used to derive the shared key k from the shared DH secret. As proof for identification and authentication, both parties sign their MAC values and exchange these signatures, optionally together with their certificates. An exact protocol flow diagram for this mode is given in Figure 13 in Appendix A.

Public Key Encryption Based Authentication. This mode requires that both parties exchanged their public keys securely beforehand (e. g. with certificates during an earlier handshake with signature based authentication). RFC 2409 advertises this mode of authentication with a plausibly deniable exchange to raise the privacy level.

In this mode, messages three and four exchange nonces and identities as ancillary information (see Figure 4). In contrast to the signature based mode, they are encrypted using the public key of the respective other party. The encoding format for these ciphertexts is PKCS #1 v1.5. For verification, both parties exchange their MAC values.

Revised Public Key Encryption Based Authentication. The PKE based mode of authentication requires both parties to perform two public- and two private-key operations. To reduce this computational overhead, the *revised* public key encryption based mode of authentication (RPKE) was invented (see Figure 8).

This mode still encrypts the nonces n_I and n_R with the other party's public key using PKCS #1 v1.5. However, the identities are encrypted with ephemeral symmetric keys ke_I and ke_R that must not be confused with k_e, which is derived later in the handshake. ke_I and ke_R are derived from each party's nonces and cookies. The rest of the handshake is identical to the non-revised mode.

PSK Based Authentication. If initiator and responder do not have asymmetric keys, symmetric PSKs can be used for authentication. This can be implemented with a (low or high entropy) password both parties know. The PSK is used to derive k from the nonces n_I and n_R, which are exchanged as ancillary information (Figure 12). The rest of the handshake is identical to the public key encryption based modes.

2.3 IKEv2

The structure of IKEv2 [24, 25] is fundamentally different from IKEv1 (cf. Figure 2) – Phase 1 and Phase 2 are partially interleaved, and Phase 2 is reduced to a two-message protocol. For our analysis it is only important that IKEv2 (cf. Figure 6) shares two authentication methods with IKEv1, and that we can directly apply our attacks to impersonate an IPsec device in Phase 1 of IKEv2.

3 Bleichenbacher Oracles

Bleichenbacher's attack is a padding oracle attack against RSA PKCS #1 v1.5 encryption padding, which is explained in more detail in Appendix B. If an implementation allows an attacker to determine if the plaintext of a chosen RSA ciphertext starts with the two bytes `0x00 0x02`, then a Bleichenbacher attack is possible. In his seminal work [9], Bleichenbacher demonstrated how such an oracle could be exploited:

Basic Algorithm. In the most simple attack scenario, attackers have eavesdropped a valid PKCS #1 v1.5 ciphertext c_0. To get the plain message m_0, the attackers issue queries to the Bleichenbacher oracle $\mathcal{O}$:

$$\mathcal{O}(c) = \begin{cases} 1 & \text{if } m = c^d \bmod N \text{ starts with } \texttt{0x00 0x02} \\ 0 & \text{otherwise} \end{cases}$$

If the oracle answers with 1, the attackers know that $2B \leq m \leq 3B-1$, where $B = 2^{8(\ell_m - 2)}$ where ℓ_m is the byte-length of message m. The attackers can then take advantage of the RSA malleability and generate new candidate ciphertexts by choosing a value s and computing

$$c = (c_0 \cdot s^e) \bmod N = (m_0 \cdot s)^e \bmod N.$$

The attackers query the oracle with c. If the oracle responds with 0, they increment s and repeat the previous step. Otherwise, the attackers learn that $2B \leq m_0 \cdot s - rN < 3B$ for some r. This allows the attackers to reduce the range of possible solutions to:

$$\frac{2B + rN}{s} \leq m_0 < \frac{3B + rN}{s}$$

The attackers proceed by refining guesses for s- and r-values and successively decreasing the size of the interval containing m_0. At some point, the interval will contain a single valid value, m_0. Bleichenbacher's original paper [9] describes this process in further detail.

Signature Forgery Using Bleichenbacher's Attack. It is well known that in the case of RSA, performing a decryption and creating a signature is mathematically the same operation. Bleichenbacher's original paper already mentioned that the attack could also be used to forge signatures over attacker-chosen data. In two papers by Jager et al. [19, 20], this has been exploited for attacks on XML-based Web Services, TLS 1.3, and Google's QUIC protocol. The ROBOT study [10] used this attack to forge a signature from Facebook's web servers as proof of exploitability.

Optimized Bleichenbacher Attack In 2012, Bardou et al. [7] presented an optimization of the standard Bleichenbacher attack by trimming the initial space for m_0. They divide a ciphertext by an integer t by multiplying it with $t^{-e} \bmod N$ with e being the public exponent of the oracle.

In case the original plaintext was divisible by t, then the multiplication $c_0 \cdot u^e \cdot t^{-e}$ is equal to $\frac{m_0}{t}$ under the assumption that m_0 and $m_0 \cdot ut^{-1}$ are PKCS #1 v1.5 conforming. Note, that the value u and t must be coprime integers with $u < \frac{2}{3}t$ and $t < \frac{2N}{9B}$.

In order to find a suitable amount of trimmer values that result in PKCS #1 v1.5 conforming messages, we need to calculate a few thousand t and u values, satisfying the above requirements. After that, we get a set of trimmer values shrinking the m_0 search space into smaller chunks of $2B \cdot \frac{t}{u} \leq m_0 < 3B \cdot \frac{t}{u}$.

4 Attack Outline

Bleichenbacher attacks [9] are adaptive chosen ciphertext attacks against RSA-PKCS #1 v1.5. Though the attack has been known for two decades, it is a common pitfall for developers [10, 27]. The mandatory use of PKCS #1 v1.5 in two ciphersuite families – the PKE (Figure 4) and RPKE (Figure 8) authentication methods – raised suspicion of whether implementations resist Bleichenbacher attacks.

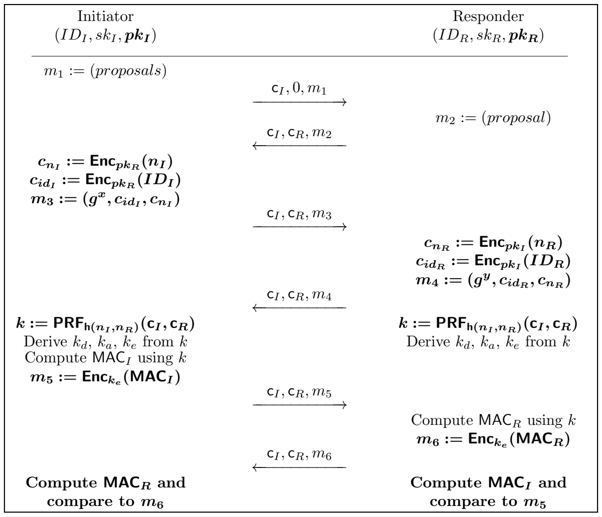
Initiator
$(ID_I, sk_I, \boldsymbol{pk_I})$
Responder
$(ID_R, sk_R, \boldsymbol{pk_R})$
$m_1 := (proposals)$
$\mathsf{c}_I, 0, m_1$
$m_2 := (proposal)$
$\mathsf{c}_I, \mathsf{c}_R, m_2$
$c_{n_I} := \mathsf{Enc}_{pk_R}(n_I)$
$c_{id_I} := \mathsf{Enc}_{pk_R}(ID_I)$
$m_3 := (g^x, c_{id_I}, c_{n_I})$
$\mathsf{c}_I, \mathsf{c}_R, m_3$
$c_{n_R} := \mathsf{Enc}_{pk_I}(n_R)$
$c_{id_R} := \mathsf{Enc}_{pk_I}(ID_R)$
$m_4 := (g^y, c_{id_R}, c_{n_R})$
$\mathsf{c}_I, \mathsf{c}_R, m_4$
$k := \mathsf{PRF}_{\mathsf{h}(n_I, n_R)}(\mathsf{c}_I, \mathsf{c}_R)$
Derive k_d, k_a, k_e from k
Compute MAC_I using k
$m_5 := \mathsf{Enc}_{k_e}(\mathsf{MAC}_I)$
$k := \mathsf{PRF}_{\mathsf{h}(n_I, n_R)}(\mathsf{c}_I, \mathsf{c}_R)$
Derive k_d, k_a, k_e from k
$\mathsf{c}_I, \mathsf{c}_R, m_5$
Compute MAC_R using k
$m_6 := \mathsf{Enc}_{k_e}(\mathsf{MAC}_R)$
$\mathsf{c}_I, \mathsf{c}_R, m_6$
Compute MAC_R and compare to m_6
Compute MAC_I and compare to m_5

Figure 4: IKEv1 in Phase 1 using main mode with PKE based authentication. Differences to Figure 3 are highlighted.

4.1 Bleichenbacher Oracles in IKEv1

PKE authentication is available and fully functional in Cisco's IOS (Internetwork Operating System). In Clavister's cOS and ZyXEL's ZyWALL USGs (Unified Security Gateways), PKE is not officially available. There is no documentation and no configuration option for it; therefore, it is not fully functional. Nevertheless, these implementations processed messages using PKE authentication in our tests. RPKE is implemented in certain Huawei devices including the Secospace USG2000 series. We were able to confirm the existence of Bleichenbacher oracles in all these implementations (CVE-2018-0131, CVE-2017-17305, CVE-2018-8753, and CVE-2018-9129), which are explained in depth in sections 5 – 7.

On an abstract level, these oracles work as follows: If we replace the ciphertext c_{n_I} in message m_3 (cf. Figure 4) with our modified RSA ciphertext, the responder will

Case 0 indicate an error (Cisco, Clavister, and ZyXEL) or silently abort (Huawei) if the ciphertext *is not* PKCS #1 v1.5 compliant, or

Case 1 continue with message m_4 (Cisco and Huawei) or return an error notification with a different message (Clavister and ZyXEL) if the ciphertext *is* PKCS #1 v1.5 compliant.

Each time we get a Case 1 answer, we can advance the Bleichenbacher attack one more step.

If a Bleichenbacher oracle is discovered in a TLS implementation, then TLS-RSA is broken since one can compute the *Premaster Secret* and the TLS session keys without any time limit on the usage of the oracle. For IKEv1, the situation is more difficult: Even if there is a strong Bleichenbacher oracle in PKE and RPKE mode, our attack must succeed within the lifetime of the IKEv1 Phase 1 session, since a DHKE during the handshake provides an additional layer of security that is not present in TLS-RSA. For example, for Cisco this time limit is

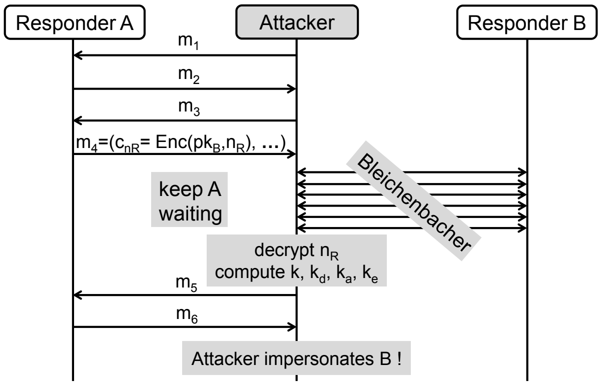

Figure 5: Bleichenbacher attack against IKEv1 PKE based authentication.

currently fixed to 60 seconds for IKEv1 and 240 seconds for IKEv2.

To phrase it differently: In TLS-RSA, a Bleichenbacher oracle allows to perform an *ex post* attack to break the confidentiality of the TLS session later on, whereas in IKEv1 a Bleichenbacher oracle only can be used to perform an *online* attack to impersonate one of the two parties in real time.

4.2 A Bleichenbacher Attack against PKE and RPKE

Figure 5 depicts a direct attack on IKEv1 PKE:

1. The attackers initiate an IKEv1 PKE based key exchange with Responder A and adhere to the protocol until receiving message m_4. They extract c_{n_R} from this message, and record the public values $\mathsf{c}_I, \mathsf{c}_R$. They also record the nonce n_I and the private DHKE key x chosen by themselves.
2. The attackers keep the IKE handshake with Responder A alive for a maximum period $t_{timeout}$. For Cisco and ZyXEL, we know that $t_{timeout} \geq 60s$, for Clavister and Huawei $t_{timeout} \geq 30s$.
3. The attackers initiate several parallel PKE based key exchanges to Responder B.
   - In each of these exchanges, they send and receive the first two messages according to the protocol specification.
   - In message m_3, they include a modified version of c_{n_I} according to the Bleichenbacher attack methodology.
   - They wait until they receive an answer m_4 (Case 1), or they can reliably determine that this message will not be sent (timeout or reception of a repeated message m_2).
4. After receiving enough Case 1 answers from Responder B, the attackers compute n_R. From the DHKE share of Responder A and their private DHKE share x they compute g^{xy}.
5. The attackers now have all the information to complete the key derivation described in Table 1. They can compute MAC_I and encrypt message m_5 to Responder A with key k_e. They thus can impersonate Responder B to Responder A.

It is important to note that this attack also can be used to execute a man-in-the-middle attack against two parties. For that, the connection is interrupted by the attackers and on the following attempt to restart the IKEv1 session with a handshake, the attackers execute a Bleichenbacher decryption attack against each party. In case of success, they can decrypt and manipulate the whole traffic.

4.3 Key Reuse

Each theoretical description of some public key primitive starts with something like $(pk,sk) \xleftarrow{\$} KeyGen(1^\kappa)$ to indicate that freshly generated keys should be used if the security proof should remain valid. In practice, this is difficult to achieve. TLS now has four versions (not counting the completely broken SSL 2.0 and 3.0), three major handshake families, both prime order and elliptic curve groups, and many minor variants described in the different ciphersuites. It is practically impossible to maintain a separate key pair for each ciphersuite. Typically, a single RSA key pair together with an encryption & signing certificate is used to configure a TLS server. As a result, cross-ciphersuite [26] and cross-version [20] attacks have been shown, despite security proofs for single ciphersuite families.

For IKE, there is a similar situation: Maintaining individual key pairs for all “ciphersuite families” and versions of IKE is practically impossible and oftentimes not supported. This is the case with the implementations by Clavister and ZyXEL, for example. Thus, it is common practice to have only one RSA key pair for the whole IKE protocol family. The actual security of the protocol family in this case crucially depends on its cross-ciphersuite and cross-version security. In fact, our Huawei test device reuses its RSA key pair even for SSH host identification, which further exposes this key pair.

4.4 A Bleichenbacher Attack Against Digital Signature Based Authentication

The attack against IKEv2 with signature based authentication proceeds as follows (cf. Figures 6 and 7). It can easily be adapted to IKEv1.

1. The attackers initiate an IKEv2 *signature* based key exchange with Responder A and adhere to the pro-

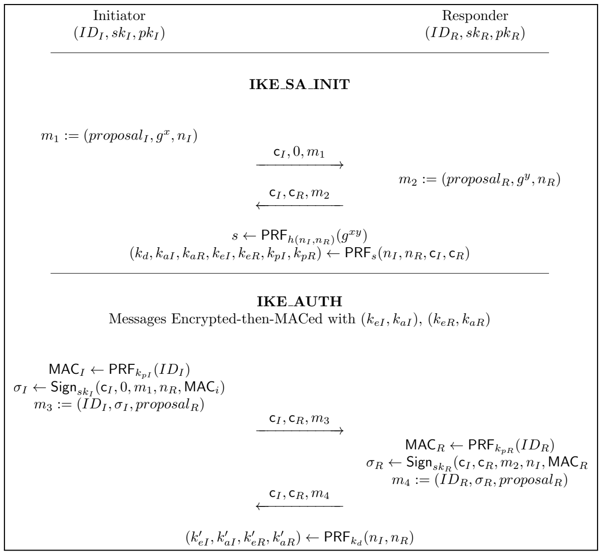
Initiator
(ID_I, sk_I, pk_I)
Responder
(ID_R, sk_R, pk_R)
IKE_SA_INIT
$m_1 := (proposal_I, g^x, n_I)$
$\mathsf{c}_I, 0, m_1$
$m_2 := (proposal_R, g^y, n_R)$
$\mathsf{c}_I, \mathsf{c}_R, m_2$
$s \leftarrow \mathsf{PRF}_{h(n_I,n_R)}(g^{xy})$
$(k_d, k_{aI}, k_{aR}, k_{eI}, k_{eR}, k_{pI}, k_{pR}) \leftarrow \mathsf{PRF}_s(n_I, n_R, \mathsf{c}_I, \mathsf{c}_R)$
IKE_AUTH
Messages Encrypted-then-MACed with (k_{eI}, k_{aI}), (k_{eR}, k_{aR})
$\mathsf{MAC}_I \leftarrow \mathsf{PRF}_{k_{pI}}(ID_I)$
$\sigma_I \leftarrow \mathsf{Sign}_{sk_I}(\mathsf{c}_I, 0, m_1, n_R, \mathsf{MAC}_i)$
$m_3 := (ID_I, \sigma_I, proposal_R)$
$\mathsf{c}_I, \mathsf{c}_R, m_3$
$\mathsf{MAC}_R \leftarrow \mathsf{PRF}_{k_{pR}}(ID_R)$
$\sigma_R \leftarrow \mathsf{Sign}_{sk_R}(\mathsf{c}_I, \mathsf{c}_R, m_2, n_I, \mathsf{MAC}_R$
$m_4 := (ID_R, \sigma_R, proposal_R)$
$\mathsf{c}_I, \mathsf{c}_R, m_4$
$(k'_{eI}, k'_{aI}, k'_{eR}, k'_{aR}) \leftarrow \mathsf{PRF}_{k_d}(n_I, n_R)$

Figure 6: IKEv2 with interleaved Phase 1/Phase 2 with signature based authentication.

tocol until they receive message m_2. After this message, they have enough data to complete the key derivation described in Figure 6. From these keys they need k_{pI} to compute $\mathsf{MAC}_I = \mathsf{PRF}_{k_{pI}}(ID_B)$, which is part of the data to be signed with the private key of Responder *B*.

2. They keep the IKE handshake with Responder *A* alive for a maximum period $t_{timeout}$. For Cisco IOS, we know that $t_{timeout} \geq 240s$.
3. The attackers encode the hash *h* of $(\mathsf{c}_I, 0, m_1, n_R, \mathsf{MAC}_I)$ with PKCS #1 v1.5 for digital signatures. We denote this encoded value as *H*. They then compute $c \leftarrow (H \cdot r^e) \pmod N$, which is known as the *blinding* step in the Bleichenbacher attack.
4. The attackers initiate several parallel *PKE based* key exchanges with Responder *B*.
    - In each of these exchanges, they send and receive the first two messages according to Figure 4.
    - In message m_3, they include a modified version of *c* according to the Bleichenbacher attack methodology.
    - They wait until they receive an answer m_4 (Case 1), or they can reliably determine that this message will not be sent (timeout, or reception of a repeated message m_2).
5. After receiving enough Case 1 answers from Responder *B*, the attackers can compute the decryption $m \leftarrow c^d \pmod N$. Since $m = c^d = (H \cdot r^e)^d = H^d \cdot r^{ed} = H^d \cdot r \pmod N$, they can compute a valid signature σ of *H* by multiplying *m* with $r^{-1} \pmod N$.

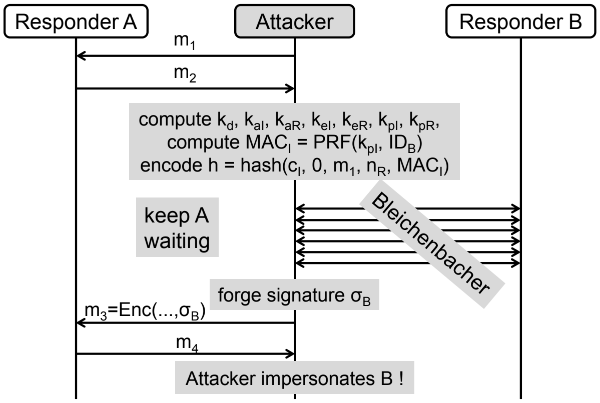
Responder A
Attacker
Responder B
m1
m2
compute kd, kaI, kaR, keI, keR, kpI, kpR,
compute MACI = PRF(kpI, IDB)
encode h = hash(cI, 0, m1, nR, MACI)
keep A waiting
Bleichenbacher
forge signature σB
m3=Enc(...,σB)
m4
Attacker impersonates B !

Figure 7: Bleichenbacher attack against IKEv2 signature based authentication.

6. The attackers complete the handshake by sending message m_3 including the valid signature σ to Responder *A*, thus impersonating Responder *B*.

4.5 Offline Dictionary Attack on Main Mode IKEv1 with Pre-Shared Keys

It is common knowledge that the aggressive mode of IKEv1 using PSKs is susceptible to offline dictionary attacks, against *passive attackers* who only eavesdrop on the IP connection. This has actually been exploited in the past [5].

We show that an offline dictionary attack is also possible against the main mode of IKEv1 and against IKEv2 with PSKs, if the attackers are *active* and interfere with DHKE. Additionally, the attackers have to act as a responder, thus waiting for a connection request by the victim initiator. Once the attackers have actively intercepted such an IKE session, they learn an encrypted MAC_I value. This value is computed from public data from the intercepted session, the shared DHKE value, and the PSK. Since the attackers know all of these values except the PSK, they can now perform an offline dictionary attack against it. Details on this attack can be found in section 9.

5 Bleichenbacher Oracle in Cisco IOS

Cisco includes the PKE authentication mode in IOS, which is the operating system on the majority of Cisco routers and current Cisco switches. The mode can also be found in IOS XE, which combines a Linux kernel with IOS as an application. IOS XE is used on Cisco's carrier routers and enterprise switches [13]. For our tests, we used a Cisco ASR 1001-X router running IOS XE in version 03.16.02.S with IOS version 15.5(3)S2.

Based on the default configuration, we first generated an RSA key pair on the device using the default options (i. e., we created *general-keys*; cf. Appendix C). Second, we created a *peer entry* with the RSA public key and IP address of our test initiator. Third and last, we configured a policy that only IKEv1 and only PKE authentication is allowed. Our test initiator is based on *Scapy* [8], a Python library for network packet manipulation. With it, we can create any IKE message and fully control all fields like cookies, proposals, nonces, ciphertexts, etc.

Ciphertext c_{n_I} in Figure 4 is the target of our attack. This ciphertext is sent with message m_3 of an IKEv1 handshake. After sending an invalid ciphertext to our Cisco router, no error message is sent back to the initiator. Instead, the router retransmits message m_2 to the initiator after one second has elapsed. If the router succeeds decrypting the message, m_4 is sent immediately to the initiator. This is clearly a Bleichenbacher oracle.

5.1 Testing the Oracle's Strength

For testing PKCS #1 v1.5 compliance, after decrypting c_{n_I}, the responder should check if the first two bytes of the plaintext are indeed `0x00 0x02`, if the following eight bytes are non-zero, and then search for the first zero byte. All data following this zero byte are considered the decrypted message.

Our test device performs all these checks after decrypting c_{n_I}. As an edge case, Cisco's implementation also accepts a plaintext that entirely consists of padding, i. e. where the zero byte separating padding and message is the last byte of the plaintext. Furthermore, IOS ignores c_{ID_I} and determines the public key to use for its response based on the IP address of the initiator. One can even omit c_{ID_I} when constructing m_3; it does not have any effect on the Bleichenbacher oracle.

This makes the Cisco oracle a *FFT* oracle based on the observations made by Bardou et al. [7]. The probability to get a valid padded message for such an *FFT* oracle is $Pr(P|A) = 0.358$ with $Pr[A] \approx 2^{-16}$ being the probability that the first two bytes are 0x0002 [7, 9]. For a 128-byte RSA modulus, the probability $Pr(P|A)$ can be computed as follows:

$$Pr(P|A) = (\tfrac{255}{256})^8 * (1 - (\tfrac{255}{256})^{118}) \approx 0.358$$

Based on the assumption made by Bleichenbacher we would need 371,843 requests for a 1024-bit modulus (128 bytes):

$$\frac{(2*2^{16}+16*128)}{Pr(P|A)} = 371,843$$

However, Bleichenbacher made his heuristic approximation based on the upper bound, not the mean value. Furthermore, we implemented the optimized Bleichenbacher attack as proposed by Bardou et al. [7], thus, we need fewer requests (247,283 on average) to mount the decryption attack.

5.2 Performance Restrictions

Oracle Performance Restrictions. In order to investigate the performance restriction we used the debug logs of Cisco IOS. There one can see that IKE handshakes are processed by a state machine. This state machine enforces some non-cryptographic boundary conditions, which have impact to the performance of a Bleichenbacher attack against Responder B. For example, IOS has a limit for concurrent SAs under negotiation of 900.

Unfortunately, Cisco's implementation is not optimized for throughput. From our observations, we assume that all cryptographic calculations for IKE are done by the device's CPU despite it having a hardware accelerator for cryptography. One can easily overload the device's CPU for several seconds with a standard PC bursting handshake messages, even with the default limit for concurrent handshakes. Moreover, even if the CPU load is kept below 100 %, we nevertheless observed packet loss. With 1024-bit RSA keys, our test device is capable of handling only 850 Bleichenbacher requests per second on average. We also saw significant CPU load after around 64,000 Bleichenbacher oracle requests, possibly caused by a memory limitation of our test device. For other devices or more powerful ones, this is probably not a limitation. Another possible reason is that hash collisions occur when the device needs to store many cookie-value pairs in its SA database due to the high amount of IKE handshakes during the attack.

Attack Performance Restrictions. For an attack, Responder A has to be held waiting. Here, a limitation in IKEv1 is the *quick mode timer*. It is started after receiving the first handshake message. If the quick mode handshake (i. e. phase 2 of the IKE handshake) is not completed after 75 seconds, this timer cancels the handshake deleting all ephemeral values like the cookie c_R, the nonce n_R, and the DH secret y.

Furthermore, the state machine maintains an error counter with a fixed limit of five. Every time an erroneous message is received or the device retransmits a message during Phase 1, the counter is incremented. Retransmissions happen every ten seconds if no message was received during that time, which we refer to as *SA timeouts*. After a fifth retransmission of any Phase 1 packet, IOS waits one last time for ten seconds before canceling the handshake. This translates to a maximum of 60 seconds between two messages sent from the peer.

For an attack, the attackers require the victim's DHKE share that is sent with message m_3 or m_4, depending on the role the attackers play. If the attackers play the role of an initiator, a Bleichenbacher attack has to be successful within the maximum of 60 seconds between messages m_4 and m_5. If the attackers play the role of a responder, a few seconds can be gained by delaying message m_4 slightly below ten seconds so that no retransmission is triggered.

In Cisco's IKEv2 implementation, timers are more relaxed. Here, an attack can take up to 240 seconds until a timeout occurs.

6 Bleichenbacher Oracles in implementations by Clavister & ZyXEL

Clavister cOS and the firmware of ZyXEL ZyWALL USGs do not officially support the PKE authentication mode. It is not documented in their manuals and the web and command line interfaces do not offer any configuration option for it. Nevertheless, both implementations responded to handshake proposals with PKE authentication in our tests. For these, we used a virtual Clavister cOS Core in version 12.00.06 and a ZyXEL ZyWALL USG 100 running firmware version 3.30 (AQQ.7).

For PKE authentication, both implementations use the key pair that is configured for IKEv1 authentication with signatures. Both implementations show the same behavior regarding the handling of IKEv1 (e. g. both respond with identical error messages).

PKE authentication with Clavister and ZyXEL is nonfunctional since one cannot configure public keys for peers. Therefore, we always expect an error notification after sending message m_3. When sending an invalid ciphertext c_{n_I} with message m_3, we receive an error message containing only 16 seemingly random bytes. A valid c_{n_I} instead triggers an error message containing the string "Data length too large for private key to decrypt". While the error message itself is misleading (the ciphertext can in fact be decrypted by the private key), the difference in the error messages is clearly a Bleichenbacher oracle.

Clavister and ZyXEL perform the same checks as Cisco. Therefore, the strength of the oracle and the estimated amount of messages is identical to the Cisco case. We did not evaluate the performance of an attack against these oracles.

7 Bleichenbacher Oracle in Huawei Secospace USG2000 series

We identified Huawei as another large network equipment supplier who offers the RPKE mode with certain devices such as their Secospace USG2000 series [18]. For our tests, we used a Huawei Secospace USG2205 BSR firewall running firmware version V300R001C10SPC700.

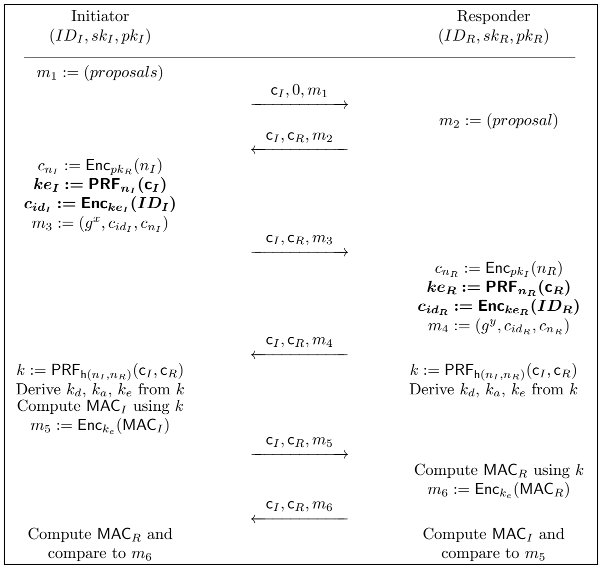
Initiator
(ID_I, sk_I, pk_I)
Responder
(ID_R, sk_R, pk_R)
$m_1 := (proposals)$
$\mathsf{c}_I, 0, m_1$
$m_2 := (proposal)$
$\mathsf{c}_I, \mathsf{c}_R, m_2$
$c_{n_I} := \mathsf{Enc}_{pk_R}(n_I)$
$\boldsymbol{ke_I := \mathsf{PRF}_{n_I}(\mathsf{c}_I)}$
$\boldsymbol{c_{id_I} := \mathsf{Enc}_{ke_I}(ID_I)}$
$m_3 := (g^x, c_{id_I}, c_{n_I})$
$\mathsf{c}_I, \mathsf{c}_R, m_3$
$c_{n_R} := \mathsf{Enc}_{pk_I}(n_R)$
$\boldsymbol{ke_R := \mathsf{PRF}_{n_R}(\mathsf{c}_R)}$
$\boldsymbol{c_{id_R} := \mathsf{Enc}_{ke_R}(ID_R)}$
$m_4 := (g^y, c_{id_R}, c_{n_R})$
$\mathsf{c}_I, \mathsf{c}_R, m_4$
$k := \mathsf{PRF}_{\mathsf{h}(n_I, n_R)}(\mathsf{c}_I, \mathsf{c}_R)$
Derive k_d, k_a, k_e from k
Compute MAC_I using k
$m_5 := \mathsf{Enc}_{k_e}(\mathsf{MAC}_I)$
$k := \mathsf{PRF}_{\mathsf{h}(n_I, n_R)}(\mathsf{c}_I, \mathsf{c}_R)$
Derive k_d, k_a, k_e from k
$\mathsf{c}_I, \mathsf{c}_R, m_5$
Compute MAC_R using k
$m_6 := \mathsf{Enc}_{k_e}(\mathsf{MAC}_R)$
$\mathsf{c}_I, \mathsf{c}_R, m_6$
Compute MAC_R and compare to m_6
Compute MAC_I and compare to m_5

Figure 8: IKEv1 in Phase 1 using main mode with RPKE based authentication. Differences to Figure 4 are highlighted.

The steps for setting up an IPsec configuration are very similar to Cisco. We started with the default configuration and generated an RSA key pair. Importing the RSA public key of our *Scapy* based test initiator turned out to be a little more complicated since the required data format is non-standard. Similar to Cisco, we configured a proposal, a policy, and a policy-template so that only IKEv1 with RPKE authentication is allowed.

Again, ciphertext c_{n_I} (cf. Figure 8) is the target of our attack. After sending an invalid ciphertext with m_3 to the device, the firewall does not send an error message back to the initiator. In contrast to Cisco's implementation, there are no retransmissions. If the firewall succeeds in processing the message, m_4 is sent to the initiator. This is also clearly a Bleichenbacher oracle.

7.1 Testing the Oracle's Strength

Huawei's firewall also performs all PKCS #1 v1.5 checks mentioned in subsection 5.1 after decrypting c_{n_I}. Therefore, Huawei's oracle is similar to the *FFT* oracle.

However, the constraints of the RPKE mode reduce the strength of the oracle. If all PKCS #1 v1.5 checks were successful, the ephemeral key ke_I is derived and used to decrypt the identity payload c_{ID_I} in order to determine the public key to use for its response. Unfortunately, during a Bleichenbacher attack the attackers do not know which ke_I is derived. There is no way for attackers to distinguish a failed PKCS #1 v1.5 check from a failed decryption of c_{ID_I}. This reduces the probability to get a Case 1 answer from Huawei by the factor $\frac{112}{256}$. Thus, Huawei's Bleichenbacher oracle has an additional false negative rate of 56.64 %, which is explained in more detail in the next section. Consequently, we estimate that a successful attack requires $371,843/(1-0.5664) = 857,571$ requests.

7.2 Oracle Performance Restrictions

RFC 2409 defines an unusual padding for messages encrypted using symmetric algorithms: The message is padded with zero bytes. The last padding byte contains the number of zero bytes inserted. Padding is mandatory even if this requires an additional block containing only padding. Figure 9 gives examples of this padding.

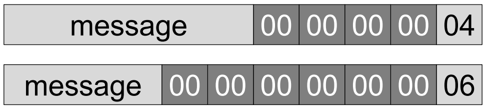

Figure 9: The padding scheme for symmetric encryptions defined by RFC 2409.

Huawei's implementation of this padding is odd: There are no checks whether the padding bytes are in fact zero-bytes. The implementation only reads the last byte and removes the given number of bytes together with the padding length byte. It does not verify whether the value of the padding length byte is larger than the block length of the negotiated algorithm. It only cancels processing if the value of the padding length byte is larger than the decrypted ciphertext or if the padding length byte is zero.

In contrast to Cisco, we observed that the Huawei device as responder thoroughly checks the identity payload c_{ID_I} sent by the initiator. It has to be present, its length has to be a multiple of the symmetric algorithm's block length, and the plaintext needs to be correctly padded in terms of the checks described above. If the plaintext identity ID_I after removing the padding is 121 or less bytes in length, the device however ignores the identity value and continues the handshake using the initiator's configured public key based on its IP address. If ID_I is 122 bytes long, the device crashes and reboots, which takes several minutes. If ID_I is 123 to 255 bytes long, ID_I is used to determine the public key of the initiator. If ID_I is more than 256 bytes long, the Huawei device also crashes and reboots.

This complicates a Bleichenbacher attack scenario: Even if the attackers hit a PKCS #1 v1.5 compliant message, the decrypted value (i. e. what the device treats as the nonce n_I) is unknown to them. This value is then used to derive the key ke_I, which in turn is used to decrypt c_{ID_I} supplied by the attackers. Since the attackers do not have ke_I, they cannot construct any c_{ID_I} that decrypts to a meaningful ID_I. During our tests, we sent random bytes for c_{ID_I} to our test device. However, even without influence on ID_I, the attackers can adjust the length of c_{ID_I}.

Here, the attackers have to deal with two contradicting restrictions: On the one hand, it is necessary to keep the length of ID_I below 122 bytes to prevent both a crash and the evaluation of the value of ID_I. On the other hand, no assumptions on the padding length byte can be made. The longer the length of c_{ID_I}, the higher the possibility that the value of the padding length byte is below the plaintext length so that no padding error occurs.

Regardless of the length of c_{ID_I}, the padding length byte can only decrypt to one of 256 possible values. Taking into account that the length of c_{ID_I} has to be a multiple of 16 (the block length of AES), the attackers have to choose between a c_{ID_I} with a length of 128 bytes and one with 112 bytes. For 128 bytes, all padding length byte values above 121 and zero will make the device not respond, either due to a padding error, an evaluation of ID_I, or a crash. This way, the Bleichenbacher oracle has an additional false negative rate of 47.66 %.

For 112 bytes, the chance of getting a Case 1 answer is slightly lower. Now, all padding length byte values above 111 and zero will make the device not respond due to the padding error. With this choice, the Bleichenbacher oracle has an additional false negative rate of 56.64 %. However, this choice eliminates the chance of hitting the crash condition with 122 bytes. Therefore, we recommend a length of 112 bytes for c_{ID_I}, which favors reliability of the attack over speed.

8 Implementing Bleichenbacher Attacks

For our proof-of-concept attack, we focused on our Cisco test device due to the high false negative rate of the Huawei oracle. In order to keep the required time for an attack below the limits, we built a highly parallelized Bleichenbacher attacker using Java (cf. Figure 10). This tool pipelines all steps of the attack through *IN* and *OUT* queues and keeps track of used and unused SAs.

SA States. There is a global limit of 900 Phase 1 SAs under negotiation per Cisco device in the default configuration. If this number is exceeded, one is blocked. Thus, one cannot start individual handshakes for each Bleichenbacher request to issue. Instead, SAs have to be reused as long as their error counter allows.

For that, we are pooling SAs and tracking their states. This is necessary since for example receiving a message m_2 can have three meanings: (1.) The SA has been created as a response to a message m_1, (2.) a Bleichenbacher request was not successful and message m_2 was a retransmission after one second, or (3.) the SA was not recently used for a request and message m_2 was a retransmission after ten seconds.

When preparing a Bleichenbacher request, an SA is taken from the *unused SA* pool and put into the *used SA* pool to ensure that SAs are not mixed up. In a parallel attack, constant SA state checks at all processing steps are required. After receiving a response to a Bleichenbacher request, we return the corresponding SA to the *unused SA* pool.

In our Bleichenbacher attacker, an SA can only be in one out of eight states. The life of an SA starts with the generation of an initiator cookie c_I. With it, the first message m_1 is send and the state of the SA is set to *PRESTART*. When we receive a corresponding message m_2, we store the responder cookie value for that SA and update its state to *FRESH*. From now on, every time we receive a message m_2 for that SA, we increment its state from *FIRST* to *FIFTH*. After the *FIFTH* state is reached and another timeout or Bleichenbacher response is received, we set the state to *EXHAUSTED* and remove the SA from the *unused SA* pool.

Packet and Network Pool. For a fast attack, we require an efficient packet builder and analyzer. The former only creates either first messages (m_1) for SA generation or third messages (m_3) for Bleichenbacher requests. The latter analyzes the responses from the Bleichenbacher oracle. Our packet builder uses static bytes sequences for the messages updating only the cookie values and encrypted nonce payloads. We omit the identity payload c_{ID_I} from m_3 in order to save an unnecessary public key decryption. The analyzer only needs the length of a received message and the values of two bytes at specific positions in order to distinguish Bleichenbacher responses from timeout packets.

For sending and receiving packets with multiple threads, we use *Java NIO DatagramChannels* and *NIO Selectors*.

Bleichenbacher Producer and Consumer. A special producer thread executes the Bleichenbacher attack against a target and distributes the computations to consumers. We implemented two distribution mechanism (multiple and single interval) in order to address the different steps in Bleichenbacher's attack.

The consumers do the expensive computations for the Bleichenbacher attack. In order to address the different computations in the two attack variants (standard and optimized), the consumers are provided with a task description of whether a multiplication or a division of the ciphertext is required. Other consumers are used to verify the results from the packet analyzer and to notify the producer in case a valid padding was found.

Cisco Oracle Simulator. In order to accelerate our evaluation process, we first queried our test device with different valid and invalid PKCS #1 v1.5 messages. After that, we analyzed its responses and reimplemented its behavior as a local multi-threaded simulator. Thus, the speed of finding valid PKCS #1 v1.5 messages is only limited by the hardware resources of the attackers' systems.

8.1 Evaluation of the Bleichenbacher IKEv1 Decryption Attack

For the decryption attack from subsection 4.2 on Cisco's IKEv1 responder, we need to finish the Bleichenbacher attack in 60 seconds. If the public key of our ASR 1001-X router is 1024 bits long, we measured an average of 850 responses to Bleichenbacher requests per second. Therefore, an attack must succeed with at most 51,000 Bleichenbacher requests.

Based on this result, we used our Cisco oracle simulator to measure the percentage of attacks that would succeed before the time runs out. These results can be found in Figure 11.

Standard Bleichenbacher. In total, we executed 990 decryption attacks with a 1024-bit public key and different encrypted nonces. On average, a decryption using Bleichenbacher's original algorithm requires 303,134 requests. However, in 78 simulations, we needed less than 51,000 request to decrypt the nonce and thus could have impersonated the router.

Optimized Bleichenbacher. For the optimized Bleichenbacher algorithm, we executed 200 attacks against our Cisco oracle simulator with different nonces and a 1024-bit key. On average, we gained a reduction for requests by approximately 18 % (247,283) using 3,000 trimmers for each attack. The amount of attacks that require less than 51,000 requests increases from 7.88 % to 26.20 %.

Real Cisco Hardware. For an attack against the real hardware, the limitations of Cisco's IKEv1 state machine are significant. The main obstacle is the SA management: Once the attackers negotiate several thousand SAs with the router, its SA handling becomes very slow.

We managed to perform a successful decryption attack against our ASR 1001-X router with approximately

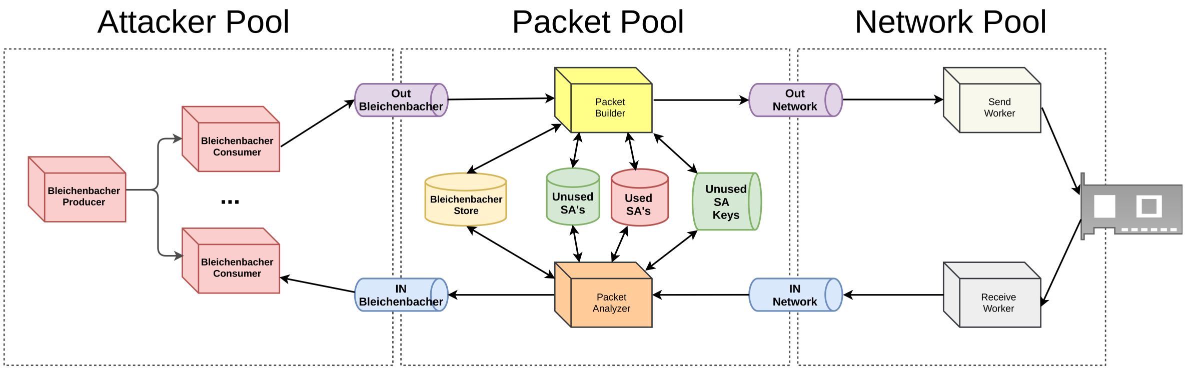

Figure 10: Design of our highly parallelized Bleichenbacher attacker.

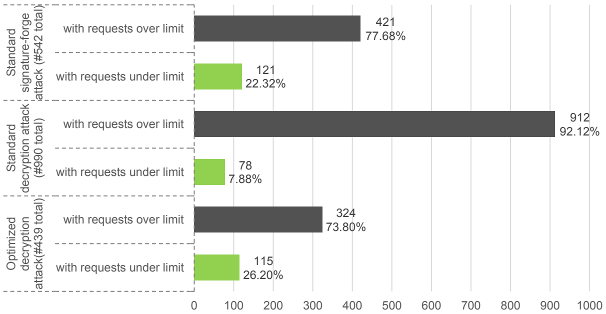

Figure 11: Statistics of 990 standard decryption, 439 optimized decryption, and 542 signature-forgery attacks against our Cisco Bleichenbacher oracle simulator.

19,000 Bleichenbacher requests. However, due to the necessary SA negotiations, the attack took 13 minutes.

Note that a too slow Bleichenbacher attack does not permanently lock out attackers. If a timeout occurs, they can just start over with a new attack using fresh values hoping to require fewer requests. If the victim has deployed multiple responders sharing one key pair (e. g. for load balancing), this could also be leveraged to speed up an attack.

8.2 Evaluation of the Bleichenbacher IKEv2 Signature Forgery Attack

For our attack with forged signatures, we have 240 seconds time. Therefore, we may issue 204,000 Bleichenbacher requests before the time runs out. The timeout limits of IKEv1 are irrelevant for this attack; the IKEv1 handshake is only used to forge the signature we need for message m_5 in IKEv2 (cf. Figure 7).

Like with the decryption attack, we used our Cisco oracle simulator in order to speed up the evaluation. We simulated 542 attacks with a 1024-bit key and random messages padded as PKCS #1 v1.5 for signatures. From these attacks, 121 signatures needed less than 204,000 Bleichenbacher requests (on average 508,520). Thus, 22 % of our attack simulations would have been fast enough to allow attackers to impersonate a Cisco router. Note that due to the increased time limit, attacking IKEv2 with a forged signature has a higher success rate than the same attack on IKEv1.

9 Offline Dictionary Attack against Weak PSKs

PSKs as authentication method are often found in scenarios where users authenticate against services such as websites and computer logins. Other applications include interconnecting devices like with Bluetooth, Wi-Fi, or IKE. In the case of IKE, knowing the PSK allows attackers to impersonate any of the peers of an IPsec connection. We will show in the following section how to mount offline dictionary attacks against IKEv1 and IKEv2.

9.1 IKEv1 with Weak Pre-Shared Keys

It is well known that the PSK based mode of authentication is vulnerable to an offline dictionary attack when used together with the aggressive mode of IKEv1 Phase 1. This has actually been exploited in the past [5]. For the main mode however, only an online attack against PSK authentication was thought to be feasible. This required attackers to initiate many handshake attempts to try all different passwords making it likely to be detected.

We present an attack that only requires a single handshake in which attackers simulate a responder. With it, the attackers learn enough information to mount an offline dictionary attack. Thus, they can learn the PSK and can thus impersonate any party or act as a *Man in the Middle*.

On the network, the attackers wait for the victim to initiate a handshake with a responder. If victim and responder already have an active connection, the attackers

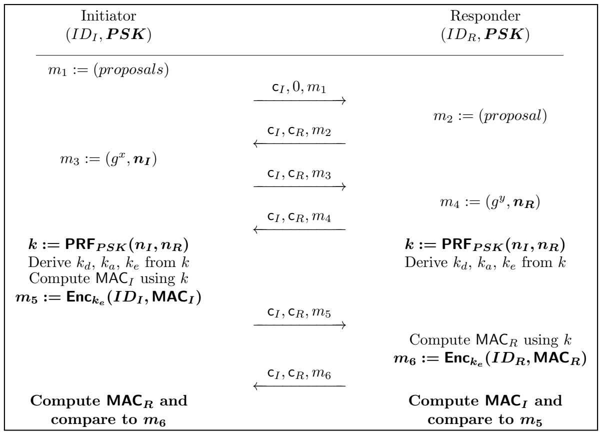
Initiator
$(ID_I, \boldsymbol{PSK})$
Responder
$(ID_R, \boldsymbol{PSK})$
$m_1 := (proposals)$
$\mathsf{c}_I, 0, m_1$
$m_2 := (proposal)$
$\mathsf{c}_I, \mathsf{c}_R, m_2$
$m_3 := (g^x, \boldsymbol{n_I})$
$\mathsf{c}_I, \mathsf{c}_R, m_3$
$m_4 := (g^y, \boldsymbol{n_R})$
$\mathsf{c}_I, \mathsf{c}_R, m_4$
$\boldsymbol{k := \mathsf{PRF}_{PSK}(n_I, n_R)}$
Derive k_d, k_a, k_e from k
Compute MAC_I using k
$\boldsymbol{m_5 := \mathsf{Enc}_{k_e}(ID_I, \mathsf{MAC}_I)}$
$\boldsymbol{k := \mathsf{PRF}_{PSK}(n_I, n_R)}$
Derive k_d, k_a, k_e from k
$\mathsf{c}_I, \mathsf{c}_R, m_5$
Compute MAC_R using k
$\boldsymbol{m_6 := \mathsf{Enc}_{k_e}(ID_R, \mathsf{MAC}_R)}$
$\mathsf{c}_I, \mathsf{c}_R, m_6$
Compute MAC_R and compare to m_6
Compute MAC_I and compare to m_5

Figure 12: IKEv1 in Phase 1 using main mode with PSK based authentication. Differences to Figure 3 are highlighted.

may enforce a new handshake by dropping all packets of the already established connection, which will eventually lead to a new handshake.

During this handshake, the attackers do not forward the packets to the responder but rather simulate to be the responder (e. g. by spoofing its IP address). The attackers act as normal responder performing the Phase 1 protocol and record all messages exchanged until they receive message m_5.

With message m_5, the attackers receive ID_I and MAC_I, encrypted with k_e (cf. Figure 12). Of all the values that m_5 is generated from, the attackers only lack knowledge of ID_I and the key k. ID_I is easy to guess, as often it is just the IP address of the initiator. The key $k = \mathsf{PRF}_{PSK}(n_I, n_R)$ is directly derived from the PSK the attackers want to learn.

This allows an offline dictionary attack against the PSK. To check whether the guessed PSK is correct, the attackers can derive k and the other three keys. If the attackers' candidate for k_e is capable of decrypting message m_5, the attack is successful and the attackers learn the PSK. This is possible since the plaintext of message m_5 has a known structure beginning with the known ID_I.

Evaluation, Impact and Countermeasure. To verify the attack, we implemented and tested it against the open source IKE implementation *strongSwan* in version 5.5.1. Since the attack solely relies on the protocol specification and does not depend on any implementation error, we believe every RFC-compliant implementation of IKEv1 to be vulnerable. Therefore, the main mode PSK authentication has to be considered as insecure as the aggressive mode one. The only available countermeasure against this attack is choosing a cryptographically strong PSK that resists dictionary attacks.

9.2 IKEv2

In general, IKEv2 is perceived to be more secure than IKEv1. However, the attack described above works similarly against IKEv2. The current standard RFC 5996 [23] mentions that it is generally not smart to rely only on a user chosen password and recommends to use IKEv2 together with EAP (Extensible Authentication Protocol) protocols. However, in practice IKEv2 is usually used without EAP.

Instead of using IKEv2 together with some EAP-TLS variant (like EAP-TTLS with EAP-MD5), one could also switch to OpenVPN and thus reduce the overhead from tunneling TLS in IKEv2. Moreover, the advice from RFC 5996 is misleading since some EAP modes like EAP-MD5 or EAP-MSCHAPv2 also do not prevent offline dictionary attacks, they just require the attackers to shift from IKE to attacking EAP. Ultimately, our research indicates that implementations only support IKEv2 with EAP for remote access of a user to a network. Site-to-site scenarios are not covered by this construction and therefore remain vulnerable to the attack.

10 Related Work

IPsec and IKE For some time, real-world cryptographic research in the area of IPsec concentrated on the encryption layer. Thus, the security of ESP is well understood today, thanks to major contributions from Paterson et al. in 2006–2007. Their work shows vulnerabilities affecting encryption-only configurations of ESP due to flaws in the standard and its implementations [14, 28]. These flaws can be resolved by integrity protection. However, in 2010 they also showed that a particular integrity protection – namely a MAC-then-encrypt configuration – also leads to a plaintext-recovery attack [15].

Research paid only little attention to IKE. The Logjam paper [5] discovered that some of the most used DH groups standardized for IKE offer an attack surface if the attackers are able to perform costly precomputations. Another contribution by Checkoway et al. shows that the random number generator used by VPN devices from Juniper Networks was manipulated leading to a passive decryption vulnerability [11]. However, both these findings do not target IKE itself, but rather the parameters of underlying cryptographic building blocks.

Bleichenbacher Attacks. Even though the seminal work by Bleichenbacher dates back to 1998 [9], Bleichenbacher vulnerabilities are discovered regularly. Though the vulnerability is not protocol-related, the majority of vulnerabilities have been found in TLS implementations. A paper by Meyer et al. found Bleichenbacher vulnerabilities in OpenSSL, JSSE (Java Secure Socket Extension), and a TLS hardware accelerator chip [27]. Somorovsky showed that MatrixSSL was also affected [29]. Recently, the ROBOT survey showed that thousands of domains on the Internet were running Bleichenbacher vulnerable servers, among them Facebook and PayPal [10].

Cross Protocol Attacks. VPNs have already been target of cross protocol attacks. One has been found in PPTP (Point-to-Point Tunneling Protocol) VPNs [17]. Another famous cross protocol attack is DROWN [6], which exploits the broken SSL 2.0 to break the current TLS 1.2. In 2012, Mavrogiannopoulos et al. described a cross-protocol attack against all TLS versions using explicit elliptic curve Diffie-Hellman parameters [26]. A paper by Jager et al. [20] shows how to attack TLS 1.3 and QUIC from a Bleichenbacher oracle in some implementation of previous TLS versions.

11 Conclusion

In this paper, we have shown that *all* versions and variants of the IPsec's Internet Key Exchange (IKE) protocol can be broken, given two entry points.

The first entry point is weak PSKs. Offline dictionary attacks are possible against all three different variants, with two different adversaries: IKEv1 PSK in aggressive mode can be broken by a passive adversary, and both IKEv1 PSK in main mode and IKEv2 PSK can be broken by an active adversary who acts as a responder.

The second entry point is Bleichenbacher oracles in the IKEv1 PKE and RPKE variants. We have shown that such oracles exist in Cisco, Clavister, Huawei, and ZyXEL devices, and have computed their strength. Given an oracle of this strength, we were able to show that under the attack restrictions imposed by Cisco's default values, we could successfully attack *all* public key-based variants of IKEv1 and IKEv2 with success probabilities between 7 % and 26 % in a single attempt. Therefore, by repeating the attacks, all implementations can be broken. In this work, we focus on IKE implementations. However, if network devices reuse RSA key pairs for other services like SSH, TLS, etc., further attack surfaces could arise.

To counter these attacks, both entry points must be closed: Only high entropy PSKs should be used, and both PKE and RPKE modes should be deactivated in all IKE devices. It is not sufficient to configure key separation on the sender side. All receivers must also be informed about this key separation – novel solutions are required to achieve this task.

Acknowledgments

The authors wish to thank Juraj Somorovsky and Tibor Jager with whom we had long conversations regarding Bleichenbacher attacks. Thanks to Cisco who provided us test hardware for our experiments. This paper is based in part upon work in the research projects *SyncEnc* and *VERTRAG*, which are funded by the German Federal Ministry of Education and Research (BMBF, FKZ: 16KIS0412K and 13N13097), as well as the *FutureTrust* project funded by the European Commission (grant 700542-Future-Trust-H2020-DS-2015-1).

Notes

[1] RFC 2409 calls these keys SKEYID, SKEYID$_d$, SKEYID$_a$, and SKEYID$_e$. We shorten these names for brevity.

[2] RFC 2409 calls these values HASH. This is misleading, since in practice the HMAC version of the negotiated hash algorithm is used as PRF. Therefore, we use the name MAC.

References

[1] Automotive Network Exchange. http://www.anx.com/.

[2] European Network Exchange. http://www.enx.com/.

[3] Japanese Network Exchange. https://www.jnx.ne.jp/.

[4] 3RD GENERATION PARTNERSHIP PROJECT (3GPP). 2018. 3GPP System Architecture Evolution (SAE); Security architecture. 3GPP TS 33.401 V15.3.0. http://www.3gpp.org/ftp/specs/archive/33_series/33.401/33401-f30.zip.

[5] ADRIAN, D., BHARGAVAN, K., DURUMERIC, Z., GAUDRY, P., GREEN, M., HALDERMAN, J. A., HENINGER, N., SPRINGALL, D., THOMÉ, E., VALENTA, L., VANDERSLOOT, B., WUSTROW, E., ZANELLA-BÉGUELIN, S., AND ZIMMERMANN, P. 2015. Imperfect forward secrecy: How Diffie-Hellman fails in practice. In *ACM CCS 15: 22nd Conference on Computer and Communications Security*.

[6] AVIRAM, N., SCHINZEL, S., SOMOROVSKY, J., HENINGER, N., DANKEL, M., STEUBE, J., VALENTA, L., ADRIAN, D., HALDERMAN, J. A., DUKHOVNI, V., KÄSPER, E., COHNEY, S., ENGELS, S., PAAR, C., AND SHAVITT, Y. 2016. DROWN: Breaking TLS with SSLv2. In *25th USENIX Security Symposium (USENIX Security 16).*

[7] BARDOU, R., FOCARDI, R., KAWAMOTO, Y., SIMIONATO, L., STEEL, G., AND TSAY, J.-K. 2012.

Efficient padding oracle attacks on cryptographic hardware. In *Advances in Cryptology–CRYPTO 2012*.

[8] BIONDI, P. Scapy. http://www.secdev.org/projects/scapy/.

[9] BLEICHENBACHER, D. 1998. Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1. In *Advances in Cryptology — CRYPTO '98*.

[10] BÖCK, H., SOMOROVSKY, J., AND YOUNG, C. 2017. Return Of Bleichenbacher's Oracle Threat (ROBOT). In *27th USENIX Security Symposium (USENIX Security 18)*.

[11] CHECKOWAY, S., MASKIEWICZ, J., GARMAN, C., FRIED, J., COHNEY, S., GREEN, M., HENINGER, N., WEINMANN, R.-P., RESCORLA, E., AND SHACHAM, H. 2016. A systematic analysis of the Juniper Dual EC incident. In *ACM CCS 16: 23rd Conference on Computer and Communications Security*.

[12] CISCO SYSTEMS INC. 2017a. Cisco ios security command reference. https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/security/a1/sec-a1-cr-book/sec-cr-c4.html#wp1444104032.

[13] CISCO SYSTEMS INC. 2017b. Configuring Internet Key Exchange for IPSec VPNs - RSA Encrypted Nonces. https://www.cisco.com/en/US/docs/ios-xml/ios/sec_conn_ikevpn/configuration/15-2mt/sec-key-exch-ipsec.html#GUID-5257C56A-122F-47F6-8BC5-3E462C946879.

[14] DEGABRIELE, J. P. AND PATERSON, K. G. 2007. Attacking the IPsec standards in encryption-only configurations. In *2007 IEEE Symposium on Security and Privacy*.

[15] DEGABRIELE, J. P. AND PATERSON, K. G. 2010. On the (In)Security of IPsec in MAC-then-encrypt configurations. In *ACM CCS 10: 17th Conference on Computer and Communications Security*.

[16] HARKINS, D. AND CARREL, D. 1998. The Internet Key Exchange (IKE). RFC 2409 (Proposed Standard). Obsoleted by RFC 4306, updated by RFC 4109.

[17] HORST, M., GROTHE, M., JAGER, T., AND SCHWENK, J. 2016. Breaking PPTP VPNs via RADIUS Encryption. In *CANS 16: 15th International Conference on Cryptology and Network Security*.

[18] HUAWEI TECHNOLOGIES CO., LTD. 2017. Authentication methods IKEv1 USG2100/2200/5100 BSR&HSR & USG2000/5000 V300R001. http://support.huawei.com/enterprise/pages/doc/subfile/docDetail.jsp?contentId=DOC1000010065&partNo=100172#authentication-method_ike_pro.

[19] JAGER, T., PATERSON, K. G., AND SOMOROVSKY, J. 2013. One bad apple: Backwards compatibility attacks on state-of-the-art cryptography. In *ISOC Network and Distributed System Security Symposium – NDSS 2013*.

[20] JAGER, T., SCHWENK, J., AND SOMOROVSKY, J. 2015. On the security of TLS 1.3 and QUIC against weaknesses in PKCS #1 v1.5 encryption. In *ACM CCS 15: 22nd Conference on Computer and Communications Security*.

[21] KALISKI, B. 1998. PKCS #1: RSA Encryption Version 1.5. RFC 2313 (Informational). Obsoleted by RFC 2437.

[22] KAUFMAN, C. 2005. Internet Key Exchange (IKEv2) Protocol. RFC 4306 (Proposed Standard). Obsoleted by RFC 5996, updated by RFC 5282.

[23] KAUFMAN, C., HOFFMAN, P., NIR, Y., AND ERONEN, P. 2010. Internet Key Exchange Protocol Version 2 (IKEv2). RFC 5996 (Proposed Standard). Obsoleted by RFC 7296, updated by RFCs 5998, 6989.

[24] KAUFMAN, C., HOFFMAN, P., NIR, Y., ERONEN, P., AND KIVINEN, T. 2014. Internet Key Exchange Protocol Version 2 (IKEv2). RFC 7296 (INTERNET STANDARD). Updated by RFCs 7427, 7670.

[25] KIVINEN, T. AND SNYDER, J. 2015. Signature Authentication in the Internet Key Exchange Version 2 (IKEv2). RFC 7427 (Proposed Standard).

[26] MAVROGIANNOPOULOS, N., VERCAUTEREN, F., VELICHKOV, V., AND PRENEEL, B. 2012. A cross-protocol attack on the TLS protocol. In *ACM CCS 12: 19th Conference on Computer and Communications Security*.

[27] MEYER, C., SOMOROVSKY, J., WEISS, E., SCHWENK, J., SCHINZEL, S., AND TEWS, E. 2014. Revisiting SSL/TLS Implementations: New Bleichenbacher Side Channels and Attacks. In *23rd USENIX Security Symposium (USENIX Security 14)*.

[28] PATERSON, K. G. AND YAU, A. K. 2006. Cryptography in theory and practice: The case of encryption in IPsec. In *Advances in Cryptology – EUROCRYPT 2006*.

[29] SOMOROVSKY, J. 2016. Systematic fuzzing and testing of TLS libraries. In *ACM CCS 16: 23rd Conference on Computer and Communications Security*.

A IKEv1 with Signature Authentication

The IKEv1 and IKEv2 signature authentication modes are similar and both target of our signature forgery attack. Supplementary to the description of the IKEv2 variant (cf. Figure 6), here we present the IKEv1 signature authentication mode in detail. Figure 13 shows the message flow for this mode.

First, the initiator creates a set of proposals consisting of algorithms, key lengths, and additional parameters and sends it with his initiator cookie to the responder. The responder selects a proposal based on his configured policies. After that, initiator and responder exchange DHKE parameters and nonces.

Both peers are now able to derive all symmetric keys. In order to confirm the keys and authenticate against each other, a MAC is computed by each party using key k from the key derivation. Subsequently, two signatures are generated by the peers: one over MAC_I and one over MAC_R. After both peers exchanged their signatures and optionally the corresponding certificates, they validate the signatures and continue with Phase 2 only if the signatures are valid.

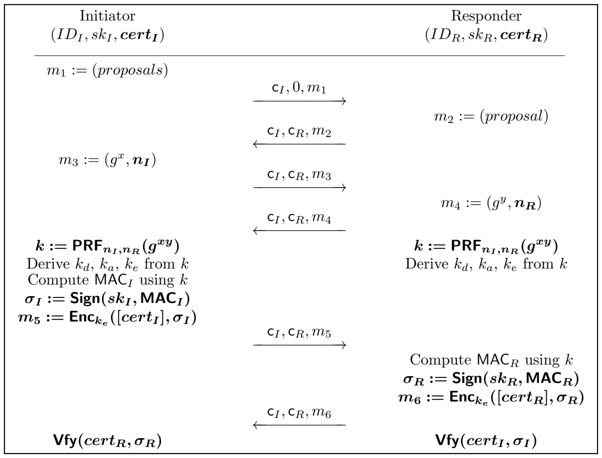
Initiator
$(ID_I, sk_I, \boldsymbol{cert_I})$
Responder
$(ID_R, sk_R, \boldsymbol{cert_R})$
$m_1 := (proposals)$
$\mathsf{c}_I, 0, m_1$
$m_2 := (proposal)$
$\mathsf{c}_I, \mathsf{c}_R, m_2$
$m_3 := (g^x, \boldsymbol{n_I})$
$\mathsf{c}_I, \mathsf{c}_R, m_3$
$m_4 := (g^y, \boldsymbol{n_R})$
$\mathsf{c}_I, \mathsf{c}_R, m_4$
$\boldsymbol{k := \mathsf{PRF}_{n_I,n_R}(g^{xy})}$
Derive k_d, k_a, k_e from k
Compute MAC_I using k
$\boldsymbol{\sigma_I := \mathsf{Sign}(sk_I, \mathsf{MAC}_I)}$
$\boldsymbol{m_5 := \mathsf{Enc}_{k_e}([cert_I], \sigma_I)}$
$\boldsymbol{k := \mathsf{PRF}_{n_I,n_R}(g^{xy})}$
Derive k_d, k_a, k_e from k
$\mathsf{c}_I, \mathsf{c}_R, m_5$
Compute MAC_R using k
$\boldsymbol{\sigma_R := \mathsf{Sign}(sk_R, \mathsf{MAC}_R)}$
$\boldsymbol{m_6 := \mathsf{Enc}_{k_e}([cert_R], \sigma_R)}$
$\mathsf{c}_I, \mathsf{c}_R, m_6$
$\boldsymbol{\mathsf{Vfy}(cert_R, \sigma_R)}$
$\boldsymbol{\mathsf{Vfy}(cert_I, \sigma_I)}$

Figure 13: IKEv1 in Phase 1 using main mode with signature based authentication. Differences to Figure 3 are highlighted.

B PKCS#1 Padding

In the following, $a||b$ denotes concatenation of strings a and b. $a[i]$ references the i-th byte in a. ℓ_a is the byte-length of string a. (N, e) denotes an RSA public key, where N is the public modulus and e is the public exponent. The corresponding secret exponent is $d = 1/e \bmod \phi(N)$.

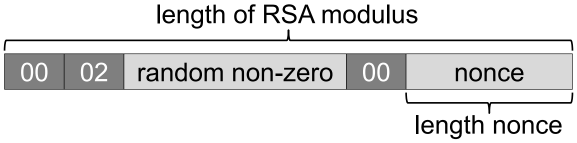

Figure 14: PKCS #1 v1.5 padding for RSA public key encryption

The PKCS #1 v1.5 encryption padding scheme [21] randomizes encryptions by requiring the encoding shown in Figure 14. To encrypt a plaintext message n (here, a nonce), the following steps have to be performed:

1. The encrypter generates a random byte string P of length $\ell_P = \ell_N - \ell_n - 3$. P must not contain 0x00 bytes (i. e. $P[i] \neq$ 0x00 $\forall i \in [1...\ell_P]$). Furthermore, P must be at least eight bytes long ($\ell_P \geq 8$).

2. The message with padding before encryption is $m =$ 0x00$||$0x02$||P||$0x00$||n$.

3. The ciphertext is computed as $c = m^e \bmod N$.

To decrypt such a ciphertext, the naïve decrypter performs the following steps:

1. Compute $m = c^d \bmod N$.

2. Check if $m[1]||m[2] =$ 0x00$||$0x02. Reject the ciphertext otherwise.

3. Check if $m[i] \neq$ 0x00 $\forall i \in [3...10]$. Reject the ciphertext otherwise.

4. Search for the first $i > 10$ such that $m[i] =$ 0x00. Reject the ciphertext if no i is found.

5. Recover the message $n = m[i+1]||...||m[\ell_N]$

However, if the attackers learn whether the decrypter rejects messages due to the checks performed in steps 2–4, the decrypter is susceptible to Bleichenbacher's attack.

C Key Types of Cisco IOS

Our key reuse attack assumes that the same RSA key pairs are used for encryption and signatures. When generating RSA key pairs, Cisco IOS gives the administrator a choice: The default is to create *general-keys*, which generates a single key pair for all authentication methods that is vulnerable to our attacks. The other option is to create *usage-keys*, through which two RSA special-usage key pairs – one encryption pair and one signature pair – are generated. In their documentation [12], Cisco states the following:

> If you plan to have both types of RSA authentication methods in your IKE policies, you may prefer to generate special-usage keys. With special-usage keys, each key is not unnecessarily exposed. (Without special-usage keys, one key is used for both authentication methods, increasing the exposure of that key.)

We have not evaluated whether special usage keys are a working countermeasure against our key reuse attack.