Applied Cryptography: Assignment 5

Group number 16

Elwin Tamminga s1013846

Lucas van der Laan s1047485

1

(a) Forward Secrecy is if the current key leaks, previously encrypted data is still secure. Backwards Secrecy on the other hand, is the opposite, which means that future messages are secure if the current key leaks.

(b) Under the condition that Diffie-Hellman uses a new key for every session and discards it after the session is over. This works well for Diffie-Hellman, because of its fast key derivation function.

(c) Yes, it provides forward secrecy, because the message keys are never used to derive any other key. This means that no other message keys should be affected.

It also has backward secrecy for the same reason, because new keys are used every single time and the keys cannot be used to derive next keys.

(d) When CK_i leaks, it does not provide backward secrecy. An attacker can compute the next chain and message keys using the key derivation function when the chain key leaks.

(e) There can be a timing issue where N is increased at the same time by Alice and Bob. This results in both sending a message with the same N, which could result in issues, depending on the implementation of the counter.

(f) Alice can decrypt c_2 to m_2 and c_3 to m_3 using the following steps:

- $SS'_1 \leftarrow B_1^{a_0} \mod p$
- $RK_1, CK_0^s \leftarrow \text{KDF}(RK_0, SS'_1)$
- $CK_1^s, MK_1 \leftarrow \text{KDF}(CK_0^s)$
- $m_2 \leftarrow \text{Dec}(MK_1, c_2)$
- $CK_2^s, MK_2 \leftarrow \text{KDF}(CK_1^s)$
- $m_3 \leftarrow \text{Dec}(MK_2, c_3)$

The roles:

- RK_i: The root key of the i-th set of chain keys. It is used to generate a new start of the chain keys.

- CK_i: The chain key of the MK_i, used to generate a new chain and message key pair.
- MK_i: The message key of the message, used to en-/decrypt the message.

(g) Alice creates a new ephemeral private key, and combines it with Bob's last public ephemeral key to compute a new shared secret. Alice needs to update the root key, using the shared secret and the current root key, and then do the chain key initialization again to obtain the new chain and message keys. After she has done this, she can now start creating messages and sending them to Bob. If she wants to send multiple messages after each other, for every message she has to update the message and chain key.

(h) i) Without the shared secret, the attacker can do literally nothing.

ii) The attacker can now finish the chain forwards and decrypt every next ciphertext that is in this part of the chain.

iii) The attacker can now decrypt the ciphertext that was encrypted using this key.

(i) The first ratchet is the symmetric-key ratchet, because the chain key is used to derive the next chain key.
The second ratchet is the Diffie-Hellman ratchet, because the shared secret is used to generate a new start of chain keys.

(j) Yes, this is less secure, as we now use the Message Key for the chain. We lose backward secrecy, because when the message key is leaked, every next message in the chain can now be decrypted as well.

2

(a) The nonce size is 12 bytes, thus 96 bits.

(b) 64 bits, because the 96-bit nonce contains 32 bits of zeros in Noise.

(c) No, because the nonce only needs to be unique per key, and in Noise the key is based on the ephemeral DH key. So a nonce of 64 does not sacrifice the security.

(d) The hash function used in WireGuard is BLAKE2s.

(e)
- The exact output size is 256 bits, as found in the WireGuard specification.
- The maximal key size is 256 bits, as found in RFC 7693.

(f) WireGuard uses the BLAKE2s hash function in a HMAC construction for keyed message authentication. However, according to RFC 7693, BLAKE2 does not require a special "HMAC" construction because it already has a built-in keying mechanism. This means that the HMAC construction can be replaced by Keyed-Blake2s.

(g) Hash functions have preimage resistance. This means that given an output hash, it is infeasable to find the key. This means that HMAC can be replaced by Keyed-Blake2s without affecting the security.

3

(a) Padding is used in the RSA PKCS#1 v1.5 block type 2, which is used for encryption. The padding string starts after the first two bytes and is pseudo-randomly generated with nonzero bytes. The length of the padding is $k - 3 - |D| \geq 8$ with $|D|$ being the length of the data block and k being the byte length of n, which is part of the public key and $n = pq$.

(b) Chosen ciphertext attack that utilizes the fact that the first two bytes are always the same if it is PKCS #1 conform. The attack can be used on a target that returns an error when the received ciphertext is not PKCS #1 conform after decrypting it. The result of the attack is the message m given the corresponding cipher text c. The ciphertext is multiplied with an arbitrary integer s^e to calculate another ciphertext. This is done multiple times, which can then be used to derive m. This only works with enough ciphertexts, according to the paper 2^{20} cipertexts is sufficient.

(c) The blinding step chooses random distinct numbers s and creates $c' = cs^e \pmod n$. It then checks if c' is PKCS #1 conform. If it is conform, then set c_i to c' and M_i to $\{[2B, 3B - 1]\}$ with $B = 2^{8(k-2)}$ and $i = i + 1$.

(d) The attacker can learn that the first two bytes of ms are 00 and 02 respectively for the chosen integer s.

(e) The probability of a positive oracle answer is $Pr(A) = \frac{B}{n}$.

(f) $m \leftarrow c^d \pmod n$ -> return true if $m[0] = 00$ and $m[1] = 02$ else return false.

(g) [1, 2, 3]
No time left to answer

References

[1] Hanno Böck, Juraj Somorovsky, and Craig Young. Return of {Bleichenbacher's} oracle threat ({{{{{ROBOT}}}}}). In *27th USENIX Security Symposium (USENIX Security 18)*, pages 817–849, 2018.

[2] A Delignat-Lavaud. Berserk vulnerability part 1 and 2: Rsa signature forgery attack due to incorrect parsing of asn. 1 encoded digestinfo in pkcs1 v1. 5 and certificate forgery in mozilla nss (technical report–intel advanced threat research team), 2014.

[3] James Manger. A chosen ciphertext attack on rsa optimal asymmetric encryption padding (oaep) as standardized in pkcs# 1 v2. 0. In *Annual international cryptology conference*, pages 230–238. Springer, 2001.