Lecture Notes for CS-578 Cryptography (SS2006) Prof. Michael Backes

5*. Secure Channels and Key Management

Lecture 8 Saarland University

5.4 Combining Privacy and Integrity

After introducing ciphers and MACs the goal now is to combine privacy and integrity. This is often called a *secure channel*, as it provides security even against active attacks. An adversary should neither be able to read any data transmitted, nor to change data without being detected.

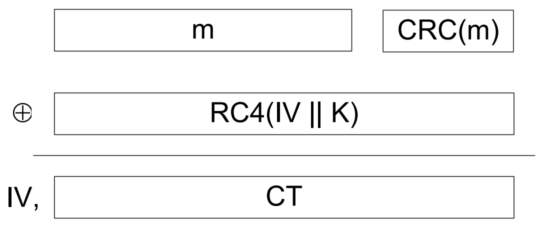

Figure 5.1: Schematic Encryption in 802.11b WEP

Let us again have a look at our favorite example, 802.11b WEP. In Figure 5.1 we sketch the relevant parts of its encryption routine. There are several weaknesses in this construction:

- Using Cyclic Redundancy Checks (CRCs) for message integrity is a poor choice, especially in the presence of active adversaries. CRC has been designed to detect errors caused by bad transmission conditions, e.g., accidentally flipped bits, not against active adversaries. First of all, it is unkeyed and, consequently, an adversary can recompute its value after changing the message. But CRC is even linear, in the sense that $\mathsf{CRC}(m \oplus B) = \mathsf{CRC}(m) \oplus \mathsf{CRC}(B)$. This can be used for the following attack against 802.11b WEP. Assume the attacker can guess which message is contained in an encryption, or at least which structure the message has. Assume he knows the message is `"My Bid is 0100$"`. Then he can easily compute an m' such that $m \oplus m' =$ `"My Bid is 0900$"`. Then he constructs $m' \mid\mid \mathsf{CRC}(m')$ and xors this to c yielding

  $$c' = m \oplus m' \mid\mid \mathsf{CRC}(m) \oplus \mathsf{CRC}(m') = m \oplus m' \mid\mid \mathsf{CRC}(m \oplus m').$$

  This is obviously a valid ciphertext for $m \oplus m'$, thus violating message integrity.

- Another weakness is the generation of the keys that are used for the RC4 stream-cipher. The IV is used as counter, and $IV \mid\mid K$ is used as key for RC4. However, RC4 is not designed to be used with related keys such as in this construction. In fact, Fluhrer, Mantin, and Shamir proved that if RC4 is used with such weak keys, then an adversary knowing only 10^6 (prefixes of) outputs of RC4 can compute the key K.

  Instead, a good method for deriving the keys would be computing $K_i := \mathsf{F}(K, IV + i)$ for a PRF F.

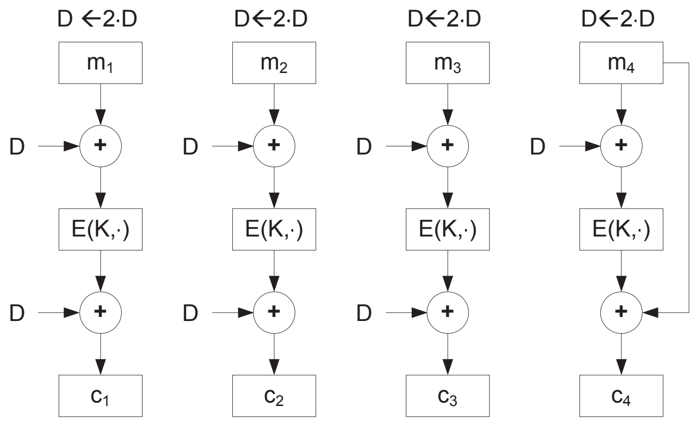

Figure 5.2: Offset Codebook Mode (OCB)

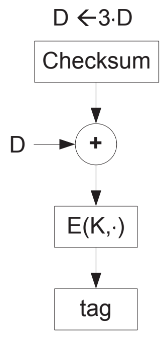

Figure 5.3: Tagging

Now let us examine more generally which possibilities exist to combine a cipher E and a MAC S, and if the resulting construction yields privacy and integrity:

- "Encrypt-then-MAC" (used in IPSec): Let $c \leftarrow \mathsf{E}(K_1, m), t \leftarrow \mathsf{S}(K_2, c)$. Then send (c, t).

  This is the recommended construction. One can prove that, for *any* CPA-secure cipher E and for *any* secure MAC S, this construction provides secrecy and integrity.

- "MAC-and-encrypt" (used in SSH v2): Let $t \leftarrow \mathsf{S}(K_1, m), c \leftarrow \mathsf{E}(K_2, m)$. Then send (c, t).

  This construction is, in general, insecure, as the definition of a MAC does not exclude, e.g., that the message is appended to the tag. However, for specific MACs such as HMAC in SSH v2, this construction is secure.

- "MAC-then-encrypt" (used in SSL): Let $t \leftarrow \mathsf{S}(K_1, m), c \leftarrow \mathsf{E}(K_2, m \mid\mid t)$. Then send c.

  This method is, in general, insecure, as, informally speaking, the tag and the encryption might interact in some bad manner. However, for specific MACs such as HMAC as in SSL, this construction is secure.

5.4.1 Offset Codebook Mode (OCB)

Constructions such as "Encrypt-then-MAC" have the disadvantage that one needs roughly $2 \cdot n$ invocations of the blockcipher for a message of length n blocks. This can be decreased to roughly $n + 1$ invocations by using OCB mode, which we will briefly describe in the sequel.

The OCB mode provides privacy and integrity at the same time. Since it is also easily parallelizable, it is easy to speed up its computation on specialized hardware. It makes use of computations in $GF(2^{128})$ to obtain different values D for the different xors in Figure 5.2. The exact description is omitted as we have not yet introduced finite fields.

Let a PRF E be given. The initialization vector IV is picked at random, and the initial D is computed as $D \leftarrow \mathsf{E}(K, IV)$. The encryption is computed according to Figure 5.2. Then a checksum is computed from these values as $\mathit{checksum} = m_1 \oplus m_2 \oplus \ldots \oplus m_{last-1} \oplus c_{last}$ and the tag is computed as in Figure 5.3. The ciphertext is $c = (IV, c_0, \ldots, c_{last}, tag)$.

The OCB mode is an optional part in 802.11i, using AES as the underlying cipher.

5.5 Key Exchange

When more than two parties want to communicate secretly with each other, they need to share multiple secret keys. There are several ways to handle these keys.

The naive approach is sketched in Figure 5.4: Each pair of participants shares a secret key which is used to encrypt every messages sent between these two parties. This has the obvious disadvantage that the number of keys each party has to store is $n-1$, so the total number of keys is quadratic in the number of participants. Furthermore, it is not clear how these keys should have been established in large networks such as the Internet.

A more effective way is using a central authority, often called *Key Distribution Center (KDC)*, as sketched in Figure 5.4. Each participant shares a secret key, and he may invoke the KDC to generate session keys if he wants to communicate with another party. This requires storage of 1 key per party: however, the KDC needs to be trusted and always be online, so it is a single point of failure. There exist several variants of this scheme, which we will not investigate any further.

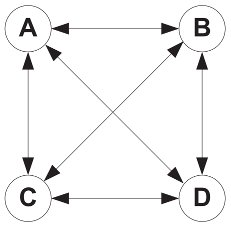

Figure 5.4: Naive Key Exchange

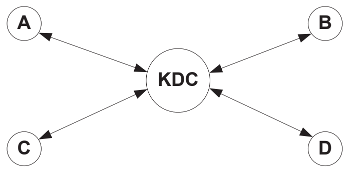

Figure 5.5: Key Exchange with a central KDC

5.5.1 Merkle Puzzles

Merkle puzzles provide a method for establishing a secret key between two parties Alice and Bob, without relying on a central authority. It was invented by Merkle in 1974 but never used in practice. Suppose Alice and Bob want to establish a shared secret key without sharing any secret before, assuming that an adversary cannot change transmitted messages, i.e., it is an passive eavesdropper. Then they can use the following method:

(1) Alice creates 2^{20} puzzles $P_i := \mathsf{E}(K_i, \text{“Puzzle } X_i\text{”} \mid\mid K_i^*)$, where K_i, X_i, K_i^* are chosen at random. The cipher E has a very short keysize, i.e., $|K_i| = 20$ bits, and $|K_i^*| = 128$ bits. She sends $(P_1, \dots, P_{2^{20}})$ to Bob.

(2) Bob picks up a random puzzle P_i, and breaks the encryption by brute-force, thus finding K_i^* and X_i. He sends X_i to Alice and keeps K_i^*.

(3) Alice can identify the correct K_i^* when receiving X_i and looking it up.

The time needed for these computations is evaluated as follows: Alice needs 2^{20} operations to encrypt the puzzles, whereas Bob needs 2^{20} operations to brute-force the puzzle he has picked up. The attacker, however, cannot relate the (randomly chosen) X_i with any puzzle P_i, so he intuitively has to brute-force all the puzzles, resulting in a runtime of $2^{20} \cdot 2^{20} = 2^{40}$.

So one can say the Merkle puzzle has a *quadratic gap*: Whereas the honest parties perform $O(n)$ steps, an adversary needs to perform $O(n^2)$ steps to recover the secret. However, such a quadratic gap is considered not to be enough for many applications. Imagine a small handheld device and an attacker owning a cluster of several fast computers. In the second half of this course we will see that public key cryptography allows for an exponential gap.