Introduction to cryptology
TD#7

2018-W15/17

Exercise 1: Block cipher key sizes

Q. 1: Let $\mathcal{E} : \{0,1\}^\kappa \times \{0,1\}^n \to \{0,1\}^n$ be a block cipher such that given enough plaintext-ciphertext pairs obtained with an unknown key, the probability of finding the key after t tries is $\approx t^{3/2}2^{-\kappa}$. Can $\mathcal{E}$ be considered to be a "good" block cipher?

Q. 2: Same question, with success probability $\approx t2^{-\kappa}$. Is it possible to have a block cipher for which this probability is significantly smaller?

Q. 3: Is it possible today for a powerful adversary to try 2^{64} keys of a block cipher? Does a key size of 64 bits offers good-enough security against such adversaries?

Q. 4: When selecting the key size of my block cipher, I set as a requirement that no (possibly powerful) adversary should be able to find a secret key with probability more than 2^{-48}. Is a key size of 96 bits enough? What about 192?

Q. 5: Suppose that an adversary is trying to find one key out of 2^{32} unknown ones. Does a key size of 96 bits offer enough security against any such adversary?

Exercise 2: LFSRs

Q. 1: What is the feedback polynomial (in $\mathbb{F}_2[X]$) of the LFSR of size 8 whose update rule is given by the following program:

```
uint8_t mul(uint8_t x)
{
        uint8_t b = (x >> 7);
        if (b)
                b = 0x1B;
        return ((x << 1) ^ b);
}
```

Q. 2: Explain how an LFSR can be used to generate an infinite sequence of bits.

Q. 2: What is the maximal possible period of the sequence generated by an LFSR of size n?

Q. 3: Recall that in CTR mode, one encrypts a message block m as $\mathcal{E}_k(ctr) \oplus m$, where ctr is a counter unique across all calls to $\mathcal{E}_k$. A typical implementation of ctr is to maintain a state initialized to the all zero string, and to increment it by one (over $\mathbb{Z}/2^n\mathbb{Z}$, where n is the block size of $\mathcal{E}$) after every call to $\mathcal{E}$.

Is it safe to replace the above counter incrementation by one stepping of a maximum-period LFSR of size n (i.e., if the current value of ctr is given by the state of the LFSR, the next value is given by the state after stepping the LFSR once)? What if the LFSR is not maximum-period?

Q. 4: Recall that in CBC mode, one encrypts a sequence of message blocks $m_0||m_1||\dots$ as: $c_0 = \mathcal{E}_k(m_0 \oplus IV)||c_1 = \mathcal{E}_k(m_1 \oplus c_0)||\dots$.

In order to generate the initial values necessary to encrypt different sequences of blocks, an implementer suggests to maintain a state intialized to a random value, used as the IV of the first block, and then to generate the next IVs by stepping a maximum-period LFSR as in the above question. Is this a good idea?

Exercise 3: Encryption with hash functions

Let $\mathcal{H} : \{0,1\}^* \to \{0,1\}^n$ be a hash function. The goal of this exercise is to design encryption systems based on such a function instead of a block cipher.

Q. 1: Recall the three security notions associated with a hash function, and for each the expected complexity (in function of n) of an optimal generic adversary that breaks it with success probability ≈ 1.

Q. 2: Let $k \xleftarrow{\$} \{0,1\}^\kappa$ be a secret key, with κ large (e.g. 256 or more). Explain why if $\mathcal{H}$ is a good hash function, it is hard to find k knowing $\mathcal{H}(k)$. Is it still hard to find k knowing $\mathcal{H}(k \oplus x)$, when x is a known κ-bit value?

Q. 3: Let m_0, m_1, k_0, k_1 be n-bit strings, where the m_i are message blocks and the k_i uniformly random (secret) keys. We want to encrypt the message a *single time* One proposes to do so as $c_0 = m_0 \oplus \mathcal{H}(k_0 \oplus m_1)$, $c_1 = m_1 \oplus \mathcal{H}(k_1 \oplus c_0)$. Explain how one can decrypt these ciphertexts, when the key is known.

Q. 4: Explain why the above scheme securely encrypts the two messages when $\mathcal{H}$ is replaced by the identity function. Do you think that the scheme is still secure when $\mathcal{H}$ is a good hash function? In which of these cases does the joint knowledge of m_0, m_1, c_0, c_1 allow to recover k_0 and k_1?

Q. 5: The above scheme only specifies how to encrypt up to $2n$ bits once. One suggests to encrypt longer messages (or equivalently, the same message more than once) $m_0, \dots, m_{2l}$ (taken to have an even number of blocks, for simplicity) as $c_0, \dots, c_{2l}$ with $c_{2i} = m_{2i} \oplus \mathcal{H}(k_0 \oplus m_{2i+1})$, $c_{2i+1} = m_{2i+1} \oplus \mathcal{H}(k_1 \oplus c_{2i})$. What is the problem of this approach? (**Xtra Bonus+++**: Give an attack that recovers k_0 and k_1 in time $2^{\kappa/2}$.)

Q. 6: One suggests to patch the approach of the previous question by defining $c_0, \dots, c_{2l}$ as $c_{2i} = m_{2i} \oplus \mathcal{H}(\varphi()||(k_0 \oplus m_{2i+1}))$, $c_{2i+1} = m_{2i+1} \oplus \mathcal{H}(\varphi()||(k_1 \oplus c_{2i}))$, where $\varphi : () \mapsto \mathsf{next}(x)$ returns the next value of a globally stored r-bit counter x initialized to zero. Explain roughly why this approach could (or does not) lead to a secure encryption scheme. In any case, what should the value of r be if one wishes to encrypt up to N message blocks in total?