Cryptology I (spring 2019) Dominique Unruh

Exercise Sheet 4

Out: 2019-04-15 Due: 2019-05-10

Problem 1: An unsavory group

Recall that ElGamal can be defined with respect to many different groups. Here we give an example of a group one should not use.

Let $p > 0$ be a large prime. Let $G := \{0, \dots, p-1\}$. The group operation is defined as follows: For $a, b \in G$, let $a \cdot b := (a + b \bmod p)$.[1] Recall that a^i for $i \in \mathbb{N}$ is defined as $a^i := a \cdot a \cdot a \cdot a \cdot a \cdot \dots \cdot a$ (i-times).

(a) What is a^i written in terms of $+$? (I.e., when unfolding the definition of the group operation, and using modular arithmetic.) This should be quite a simple operation!

(b) Show that there is an efficient algorithm for solving the discrete logarithm problem in G. That is, given $a \in G$ and $b := a^i \in G$ (but not given i), the algorithm should compute i.

Note: You can use, without proof, the fact that there is an efficient algorithm (Extended Euclidean Algorithm, EEA) that, given p and $x \in \{1, \dots, p-1\}$ computes y with $xy \equiv 1 \mod p$. (This is multiplication modulo p, not the group operation. The fact that inverting works for all x uses that p is prime.)

(c) Show that the DDH assumption does not hold for G. (I.e., there is an efficient algorithm that distinguishes the two games from the definition of the DDH assumption with probability close to 1.)

(d) Program the algorithms from (b) and (c). You can use the following template: `additive-group.py` (on the webpage)

[1] This is notationally highly confusing, of course, because it looks like we claim that plus and times are the same thing. But keep in mind that we are defining a new operation on G here, and it is just a notational convention that we write it like multiplication. In particular, do not confuse $a \cdot b$ (the group operation) with $a \cdot b \bmod p$ (actual multiplication modulo p). Often one would use the symbol $+$, but that would be confusing as well because in our definitions of ElGamal we used multiplicative notation. If you wish, you can introduce a different symbol for the group operation, say $\circ$, and then the definition becomes $a \circ b := (a + b \bmod p)$. You are free to do it either way in your solution, but make sure that you do not mix up the different meanings of $a \cdot b$!

Problem 2: Collisions in Iterated Hash

Let E be a block cipher with n-bit keys and messages. Assume the following compression function:

$$F(x\|y) := E(y, x) \oplus x.$$

(y is used as the key for E.) This is a very slight variation of the Davies-Meyer compression function.

Let H be the Iterated Hash construction using compression function F.

Assume the designer of the standard happens to have chosen the initialization vector iv as $iv := D(z, 0)$ for random z. (Here D is the decryption corresponding to E.) The designer justified this with the fact that this basically leads to a random iv.

Describe how to (efficiently) find a collision for H.

Note: You can assume that you know which z the designer used.

Hint: First explain how to efficiently construct x^* as describe in the lecture in the attack on Iterated Hash.

Problem 3: Birthday attack

Implement a birthday attack for a hash function with 48 bit output. The python code in `birthday.py` contains template code, fill in the code for the function `find_collision`.

Problem 4: MACs and encryption

Consider the following symmetric encryption scheme (KG, E, D). KG chooses an AES key. $E(k, m) := E_{AES}(k, m)\|0^{32}$. ($0^{32}$ stands for a string consisting of 32 zeros.) And the decryption $D(k, c)$ does the following: Let $c'\|p := c$ where p has length 32 bit and c' is all but the last 32 bits of c. $m := D_{AES}(k, c')$. If $p = 0^{32}$, then $D(k, c)$ returns m. If $p \neq 0^{32}$ and $k_p = 0$ (here k_p is the p-th bit of the key k), then $D(k, c)$ returns m. If $p \neq 0^{32}$ and $k_p = 1$, then $D(k, c)$ aborts.

(a) Show that (KG, E, D) can be totally broken using a chosen ciphertext attack.[2] That is, show that it is possible to recover the key k using a chosen ciphertext attack.

(b) To avoid the issue, we try to use authentication: Let MAC be an EF-CMA secure MAC. We construct a new encryption scheme E'. The key of this scheme consists of an AES key k_1 and a MAC-key k_2. Encryption is as follows: $E'(k_1k_2, m) := E(k_1, (MAC(k_2, m), m))$. Decryption D' checks the tag $MAC(k_2, m)$ and aborts if it is incorrect.[3] (This is called MAC-then-encrypt.)

[2] In a chosen ciphertext attack, the adversary is also allowed to submit plaintexts for encryption, not only ciphertexts for decryption.

[3] We assume that you cannot distinguish between an abort due to a wrong tag or an abort of the underlying algorithm D.

Does E' withstand chosen ciphertext attacks that reveal the whole key k_1? If yes, explain why (without proof). If no, how to attack?

(c) We try to use authentication in another way: Let MAC be an EF-CMA secure MAC. We construct a new encryption scheme E''. The key of this scheme consists of an AES key k_1 and a MAC-key k_2. Encryption is as follows: $E''(k_1k_2, m) := MAC(k_2, c)\|c$ with $c := E(k_1, m)$. Decryption D' checks the tag $MAC(k_2, c)$ and aborts if it is incorrect.[4] (This is called encrypt-then-MAC.)

Does E'' withstand chosen ciphertext attacks that reveal the whole key k_1? If yes, explain why (without proof). If no, how to attack?

Hint: One of (b), (c) is secure, the other is insecure.

Problem 5: Authentication in WEP (bonus problem)

In the WEP-protocol (used for securing Wifi, now mostly replaced by WPA), messages are "encrypted" using the following procedure: First, a key k is established between the parties A and B. (We do not care how, for the purpose of this exercise we assume that this is done securely.) Then, to transmit a message m, A chooses an initialization vector IV (we do not care how) and sends IV and $c := \mathit{keystream} \oplus (m\|\mathit{CRC}(m))$. Here *keystream* is the RC4 keystream computed from IV and k (we do not care how).

The function CRC is a so-called cyclic redundancy check, a checksum added to the WEP protocol to ensure integrity. We only give the important facts about CRC and omit a full description. Each bit of $CRC(m)$ is the XOR of some of the message bits. Which messages bits are XORed into which bit of $CRC(m)$ is publicly known. (In other words, the i-th bit of $CRC(m)$ is $\bigoplus_{j\in I_i} m_j$ for a publicly known I_i.)

An adversary intercepts the ciphertext c. He wishes to flip certain bits of the message (i.e., he wants to replace m by $m \oplus p$ for some fixed p). This can be done by flipping the corresponding bits of the ciphertext c. But then, the CRC will be incorrect, and B will reject the message after decryption! Thus the CRC *seems* to ensure integrity of the message and to avoid malleability. (This is probably why the designers of WEP added it here.)

Show that the CRC does not increase the security! That is, show how the adversary can modify the ciphertext c such that c becomes an encryption of $m \oplus p$ and such that the CRC within c is still valid (i.e., it becomes the CRC for $m \oplus p$).

Hint: Think of how the i-th bit of $CRC(m \oplus p)$ relates to the i-th bit of $CRC(m)$. (Linearity!)

[4] We assume that you cannot distinguish between an abort due to a wrong tag or an abort of the underlying algorithm D.