ETH Zürich, D-INFK
Spring 2018

Prof. Ueli Maurer
Christian Badertscher
Fabio Banfi

Cryptography Foundations
Solution Exercise 3

3.1 Key Recycling

The proposed scheme is insecure: If a message m known by the attacker is sent and the attacker obtains the ciphertext $c = m \oplus k$ and the tag $z = f(c, k)$, she can compute the key $k = m \oplus c$. This allows the attacker to encrypt and authenticate any message $m' \in \{0,1\}^n$ and hence break authenticity.

3.2 Combining Different Schemes

a) The security definition for MACs only guarantees unforgeability but not confidentiality. Concretely, for any given unforgeable MAC one can define a new MAC by appending the plaintext to the tag. The resulting MAC is still unforgeable, but the Encrypt-and-MAC paradigm using this MAC always leaks the plaintext to the adversary.

A MAC scheme in this context must additionally fulfill a confidentiality property, which could be defined analogously to the definitions for encryption schemes.

b) The potential connection reset can be used to detect whether or not a modification of the ciphertext also caused a modification of the plaintext, since the verification will (at least with very high probability) only succeed for the correct plaintext. This can be exploited if the probability that a certain modification of the ciphertext changes the contained plaintext depends on the actual value of the plaintext.

Consider the following encryption scheme: A given message $m = m_0 m_1 \dots m_n \in \{0,1\}^n$ is encoded using two bits via $0 \mapsto 00$, 01, or 10 (choosing any one of the three at random) and $1 \mapsto 11$. The resulting $2n$-bit string is then encrypted using the one-time pad. This encryption scheme is secure, since the one-time pad is secure.

Flipping the first bit of the ciphertext has the following effect:

- If $m_0 = 0$, then the first two bits of the encoded string are 00, 01, or 10. If the first bit of the ciphertext is flipped, one obtains a one-time pad encryption of 10, 11, or 00, respectively. Hence, the first bit of the plaintext will be recovered as 0 with probability $\frac{2}{3}$ and as 1 with probability $\frac{1}{3}$. Hence, a connection reset will occur only with probability $\frac{1}{3}$.
- If $m_0 = 1$, then the first two bits of the encoded string are 11, and flipping the first bit of the ciphertext results in a one-time pad encryption of 01. This value is decoded to 0, so the connection reset will occur (essentially) always.

This difference in probability can be used to obtain information about the first bit of the message.

3.3 Computing Discrete Logarithms in Specific Groups

a) Since $G = \mathbb{Z}_n$ is additive, the discrete logarithm of y to the base g actually is the integer $x \in \{0, \ldots, n-1\}$ such that x-times g is equal to y, which means $y = xg = g + \cdots + g$ (x times). Now $\mathbb{Z}_n$ is not only a group but also a ring. That is, we do not only have addition, but also multiplication modulo n. We can interpret x as an element of $\mathbb{Z}_n$ and write $y = xg$ where the multiplication on the right now is multiplication in the ring $\mathbb{Z}_n$. This allows us to find x by dividing by g. More precisely, that g is a generator means that g must have a multiplicative inverse, as otherwise no multiple of g would be equal to 1. Notice that the multiples of g exhaust all elements in G by the definition of a generator. We can efficiently find the multiplicative inverse g^{-1} with the help of the extended Euclidean algorithm, and with it we can easily compute $yg^{-1} = xgg^{-1} = x$.

b) We use our algorithm to compute the discrete logarithm a of h to the base g, so that $h = g^a$. Since h also is a generator, there exists a b such that $g = h^b$. Combining these two equations gives $g = h^b = g^{ab}$. Hence, b is the multiplicative inverse of a modulo n, i.e., $ab \equiv 1 \pmod{n}$. We can compute b efficiently with the extended Euclidean algorithm. Now, for an arbitrary $y \in G$, we can first compute the discrete logarithm x to the base g such that $y = g^x$. Then, $y = g^x = h^{bx}$, so bx is the discrete logarithm of y to the base h.

c) We write $x = 2q + r$ with $r = x \bmod 2 \in \{0, 1\}$ and observe that

$$y^{n/2} = y^m = g^{xm} = g^{(2q+r)m} = g^{qn} g^{rm} = g^{rm}$$

is equal to 1 precisely if x is even ($r = 0$). Indeed, clearly $g^0 = 1$ and $g^m \neq 1$, because n is the smallest non-zero exponent such that $g^n = 1$ by the definition of a generator. So, by computing y^m and comparing it to 1 we can determine if x is even or odd. Note that the square-and-multiply algorithm allows for efficient computation of y^m.

d) We want to iteratively compute the bits x_i, $i = 0, \ldots, k-1$, such that

$$x_{k-1}2^{k-1} + \cdots + x_1 2 + x_0 \equiv x \pmod{2^k}.$$

We first compute $x_0 = x \bmod 2$ as in subtask **c)**. Then we set $y_1 = y/g^{x_0}$ and compute

$$y_1^{n/4} = \frac{y^{n/4}}{g^{x_0 n/4}} = \frac{g^{xn/4}}{g^{x_0 n/4}} = \frac{g^{(x_1 2 + x_0)n/4}}{g^{x_0 n/4}} = g^{x_1 n/2},$$

where we have used in the third step that $x \bmod 4 = x_1 2 + x_0$ and $g^{4l \cdot n/4} = 1$ for $l \in \mathbb{N}$. As in subtask **c)**, $g^{x_1 n/2} = 1$ if and only if $x_1 = 0$. We proceed iteratively for $i = 2, \ldots, k-1$, set $y_i = y_{i-1}/g^{x_{i-1}2^{i-1}}$, and compute

$$y_i^{n/2^{i+1}} = \frac{y^{n/2^{i+1}}}{g^{(x_{i-1}2^{i-1} + \cdots + x_0)n/2^{i+1}}} = \frac{g^{(x_i 2^i + \cdots + x_0)n/2^{i+1}}}{g^{(x_{i-1}2^{i-1} + \cdots + x_0)n/2^{i+1}}} = g^{x_i n/2},$$

which again is equal to 1 if and only if $x_i = 0$.

e) We generalize the idea of subtask **d)** to the prime divisor p of n. We want to iteratively compute $x_i \in \{0, \ldots, p-1\}$ such that

$$x_{k-1}p^{k-1} + \cdots + x_1 p + x_0 \equiv x \pmod{p^k}.$$

Suppose we already have $x_0, \ldots, x_{i-1}$. Then, with $y_i = y/g^{x_{i-1}p^{i-1} + \cdots + x_0}$,

$$y_i^{n/p^{i+1}} = g^{x_i n/p}. \tag{1}$$

This expression is equal to precisely one of the p different p-th roots of unity $r_l = g^{ln/p}$, $l = 0, \ldots, p-1$, which we precompute once. So, $x_i = l$ if and only if $y_i^{n/p^{i+1}} = r_l$.

f) Let the prime factorization of n be given by

$$n = \prod_{i=1}^{N} p_i^{k_i}.$$

With the algorithm presented in subtask **e)**, we can efficiently compute $x_i = x \bmod p_i^{k_i}$ for all i. These can then be lifted with the Chinese remainder theorem to obtain $x = x \bmod n$. Remember that the Chinese remainder theorem says that there is exactly one solution modulo n of the system of congruence equations $x \equiv x_i \pmod{p_i^{k_i}}$, and provides an efficient algorithm to compute this solution x.

3.4 Resources and Converters in the Diffie-Hellman Key-Agreement

In the following let $\mathbb{G} = \langle g \rangle$ be a cyclic group with generator g and of order $q := |\mathbb{G}|$.

Alice's (initiator) Diffie-Hellman converter $\mathsf{dh_A}$
(for key space $\mathbb{G}$)

Variables $x_\mathsf{A} \in \mathbb{Z}_q \cup \{\bot\}, k \in \mathbb{G} \cup \{\bot\}$: Both initialized to $\bot$.

- On the first input init at outside interface, sample ξ uniformly at random from $\mathbb{Z}_q$, set $x_\mathsf{A} := \xi$, and output $y_\mathsf{A} := g^{x_\mathsf{A}}$ at inside sub-interface attached to $\bullet\longrightarrow$.
- On input getKey at outside interface (note that $x_A \neq \bot$ since init was already input):
  - If $k \neq \bot$ ($k \in \mathbb{G}$), output k at outside interface.
  - If $k = \bot$, output read at inside sub-interface attached to $\longleftarrow\bullet$, and then:
    - If $\bot$ is input at this inside sub-interface, output $\bot$ at outside interface.
    - If $y_\mathsf{B} \neq \bot$ ($y_\mathsf{B} \in \mathbb{G}$) is input at this inside sub-interface, set $k := y_\mathsf{B}^{x_\mathsf{A}}$ and output k at outside interface.

Bob's (responder) Diffie-Hellman converter $\mathsf{dh_B}$
(for key space $\mathbb{G}$)

Variable $k \in \mathbb{G} \cup \{\bot\}$: Initialized to $\bot$.

- On input getKey at outside interface:
  - If $k \neq \bot$ ($k \in \mathbb{G}$), output k at outside interface.
  - If $k = \bot$, output read at inside sub-interface attached to $\bullet\longrightarrow$, and then:
    - If $\bot$ is input at this inside sub-interface, output $\bot$ at outside interface.
    - If $y_\mathsf{A} \neq \bot$ ($y_\mathsf{A} \in \mathbb{G}$) is input at this inside sub-interface, sample x_B uniformly at random from $\mathbb{Z}_q$, and then:
      1. Output $y_\mathsf{B} := g^{x_\mathsf{B}}$ at inside sub-interface attached to $\longleftarrow\bullet$.
      2. Set $k := y_\mathsf{A}^{x_\mathsf{B}}$, and output k at outside interface.

The Diffie-Hellman shared secret key ⟷

(for key space $\mathbb{G}$)

Variables $k \in \mathbb{G} \cup \{\bot\}, s_{\mathsf{A}}, s_{\mathsf{B}} \in \{\bot, \top\}$**:** All initialized to $\bot$.

- **Interface** A**:**
  - On the first input init, set $s_{\mathsf{A}} := \top$, sample κ uniformly at random from $\mathbb{G}$, and set $k := \kappa$.
  - On input getKey, output k.
- **Interface** B**:** On input getKey, set $s_{\mathsf{B}} := \top$ and output k.
- **Interface** E**:**
  - On input $\mathsf{read}_{\mathsf{A}}$, output s_{A}.
  - On input $\mathsf{read}_{\mathsf{B}}$, output s_{B}.

Note that the flags s_{A} and s_{B} provided by the shared secret key ⟷ are necessary for the simulator to emulate at the right time the fact that the public values y_{A} and y_{B} are available on the respective authenticated channels.