Université Libre de Bruxelles
Faculté des Sciences

January 20, 2021

INFO-F-405 Introduction to cryptography

This assessment may be done with the use of personal notes, slides, books, web pages, etc. However, the assessment is strictly personal and you are not allowed to communicate with other students or other people during this assessment.

Question 1

(5/20) Practical aspects of cryptography Thelma and Louise recently met for the first time and would like to do business together. After their meeting, because of the COVID-19, they are forced to stay at home and can communicate only by phone or via the Internet. They wish to start discussing their business project, but want to do so confidentially. Hopefully, they already exchanged their phone numbers and email addresses, and are both knowledgeable about cryptography. However, their project is highly sensitive and they need to protect their conversation against even active adversaries, that is, people who can afford to tamper with Internet connections. We nevertheless assume that the adversaries are not going to cut their communication lines, as this would be too obvious and as they would otherwise miss all opportunities of spying on the two business women.

Sub-question A: Please explain how Thelma and Louise can set up a secure communication channel using cryptography, and how they can use it to communicate securely. If they use keys, specify what kind of keys and which key(s) belong(s) to whom. List clearly the assumptions that allow their communication to be confidential, including potential threats that you think your solution does not (completely) prevent against (if any).

Your solution can make black-box use of cryptographic building blocks such as secret-key encryption, message authentication code, authenticated encryption, hash function, public-key encryption, key agreement or signature. (There is no need to specify a particular choice of algorithms behind these building blocks.)

Sub-question B: The same story happens with Tom and Jerry. Unlike Thelma and Louise, however, the two businsess men have been friends since childhood, and they share lots of common memories. Many of these memories can be considered secret, as they are not known to anyone except themselves (or at least, they are unknown to the adversaries).

Propose a way for Tom and Jerry to set up a secure communication channel that takes advantage of these secret memories.

Question 2

(5/20) Secret-key cryptography Consider the following encryption mode called NXCTR for "Nonce-Xored CounTeR". It is a block cipher-based mode very similar to the original CTR, but the nonce is XORed into the output of the block cipher instead of being part of its input. More precisely, NXCTR works as in Algorithm 1 and is illustrated in Figure 1. However, NXCTR is flawed.

A. What is the practical advantage of NXCTR over the original CTR? And over CBC?

B. What is/are intuitively the security problem(s) of NXCTR?

C. Describe how an adversary can win the IND-CPA game against NXCTR with only practical data and time complexities.

D. How can you change NXCTR to make it IND-CPA secure? Why does it solve the problem?

Algorithm 1 The NXCTR encryption mode on top of block cipher E with block size n bits and key size m bits.

Input: secret key $K \in \mathbb{Z}_2^m$, plaintext $p \in \mathbb{Z}_2^*$ and nonce $N \in \mathbb{N}$
Output: ciphertext $c \in \mathbb{Z}_2^{|p|}$

Cut p into blocks of n bits $(p_0, p_1, p_2, \ldots, p_l)$, except for the last one that can be shorter
Generate the keystream, for $i = 0$ to l

$$k_i = E_K(\text{block}(i)) \oplus \text{block}(N)$$

(here block(x) encodes the integer x into a string in $\mathbb{Z}_2^n$)
Truncate the last keystream block k_l to the size of p_l
Compute the ciphertext $c_i = k_i \oplus p_i$, for $i = 0$ to l

---

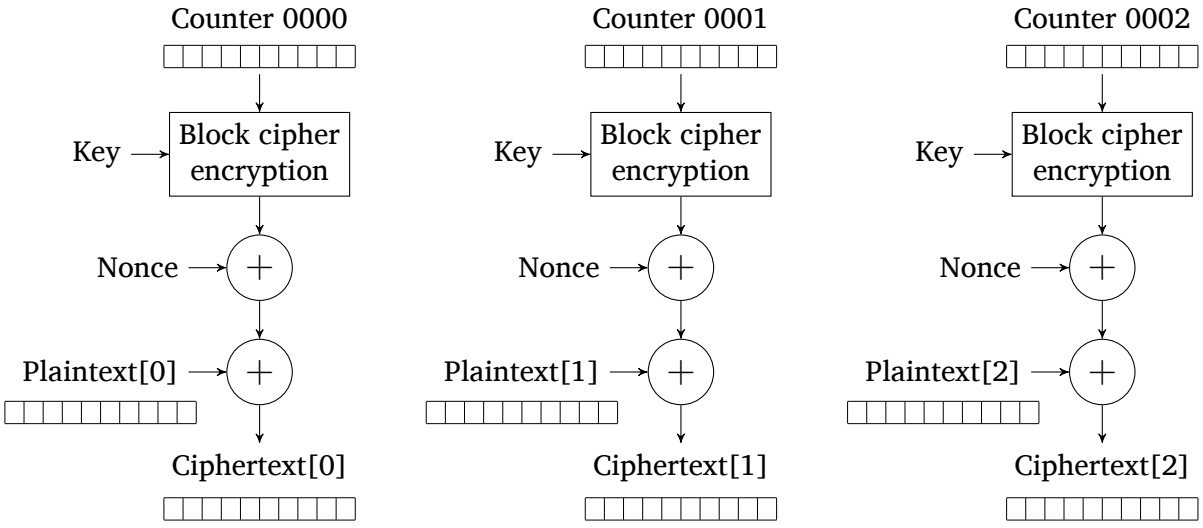

Figure 1: The NXCTR encryption mode.

Question 3

(5/20) Hashing Let π be a cryptographically secure permutation that maps $b = 1600$-bit strings onto b-bit strings. We assume that π and its inverse π^{-1} are both equally efficiently implementable. We build a hash function by applying the sponge construction on top of π. The hash function has a fixed output length of $n = 256$ bits. However, for the sake of this question, we set the capacity to $c = 0$.

In more details, for an input bit string m of any length, we proceed as follows:

- Padding: we append to m a bit 1, followed by the minimum number of bits 0 such that its length is a multiple of b bits.
- Cutting: we cut the padded input string into l blocks of b bits each: $m_1, m_2, \ldots, m_l$.
- Initialization: we set the state s to 0^{1600}, the string of 1600 zero bits.
- For each block m_i, with i running from 1 to l, we do the following:
  - We bitwise modulo-2 add (or XOR) m_i to the state: $s \leftarrow s \oplus m_i$.
  - We apply the permutation to the state: $s \leftarrow \pi(s)$.
- We return as hash the first n bits of s.

Sub-question A: Please explain how to obtain a collision with this hash function.

Sub-question B: Given two prefixes x and y of b bits each, show how to obtain a collision between one input that must start with x and the other one with y.

Sub-question C: Given an output value z of n bits, describe how to find a preimage with this hash function.

Sub-question D: Now let us assume that this hash function is used exclusively in an application that hashes input strings made of ASCII-encoded text, for which each byte has its most significant bit always set to 0. More precisely, we restrict the input strings m such that every 8th bit is 0, and for the sake of simplicity this is the only restriction we consider (i.e., we do not care whether m is actually valid ASCII-encoded text or not). Please explain how to obtain a collision with this hash function such that the colliding inputs have this restriction. What is the complexity of your attack?

Sub-question E: Given an output value z of n bits, describe how to find a preimage with this hash function such that the preimage has the aforementioned restriction. What is the complexity of your attack?

Question 4

(5/20) Public-key cryptography A company that installs and maintains an open-source operating system is creating an automatic update mechanism for its customers. Regularly, the company publishes an *update pack* containing the latest changes to be made to the operating system, and each computer connected to the Internet can fetch it. The company would like to guarantee the integrity of these update packs so as to avoid the installation of malicious software on their customers' computers. Fortunately, confidentiality is not required, as it is open-source software.

In the following, we describe a protocol that the company uses to sign the update packs and for the customers to check that the update packs are genuine. However, we voluntarily added some mistakes (including omissions) to its definition. Some of them are functional, that is, they prevent the protocol from working correctly, while others introduce security flaws.

Please *list all the mistakes* you find, and for each, *justify* why it is incorrect and *propose a correction*.

Let $\mathcal{E}$ be a standard elliptic curve over GF(p) (for some prime p) that is used by the company and its customers. They also agreed on a base point $G \in \mathcal{E}$ with prime order q, i.e., $[q]G$ is the neutral element. Note that the uppercase letters refer to points on $\mathcal{E}$, while lowercase letters are integers, and `UP` is a string of bits that represents an update pack.

One-time setup at the company:

1. The company secretly generates its secret key c randomly and uniformly in $[1 \dots p-1]$.
2. The company computes its public key $C = [c]G$, and publishes it via some PKI process. (This aspect is out of scope of this question. In the sequel, we assume that all the public keys are correctly authenticated and can be trusted.)
3. The company secretly generates its secret value n_C randomly and uniformly in $[1 \dots p-1]$.

Company's procedure to sign and post an update pack `UP`

1. Compute $k = \text{hash}(n_C)$.
2. Compute $R = [k]G$.
3. Compute $e = \text{hash}(R \| \texttt{UP})$.
4. Compute $s = k + ec \bmod p$.
5. Post $(\texttt{UP}, e, s)$ on the company's update repository.

Customer's procedure to retrieve and install an update pack

1. Retrieve $(\texttt{UP}, e, s)$ from the company's update repository.
2. Compute $C = [c]G$.
3. Compute $R' = [s]G - [e]C$.
4. Compute $e' = \text{hash}(R' \| \texttt{UP})$.
5. The customer installs `UP`.