# Security

**Assignment 4, Wednesday 1 October 2008**

**Goals**: After successfully completing this exercise you should be able to analyse simple cryptographic protocols. We also recapitulate some facts about mono-alphabetic ciphers.

**Deadline**: Friday October 10, during the lecture or earlier in Flavio's or Olha's post box (in the white cabinet near the printer at the station-side-end of the corridor on the second floor of the Huygens building).

A (3 points) In this exercise we look at the situation where Alice wants to log in to a server from her local computer. We consider a number of protocols to authenticate Alice to the server. The function $h$ used in the protocols is a cryptographic hash function.

(1) The server has a table with (name, password)-pairs. Alice enters her name and password, and her local computer sends these to the server. One could denote this as:

Server[name, password] Alice → Server : "Alice", password

(2) The server has a table with (name, hash of password)-pairs. Alice's local computer still sends her name and password to the server:

Server[name, $h$(password)] Alice → Server : "Alice", password

(3) For each name the server chooses a random number $s$, the salt, and stores the triple (name, salt, hash of the concatenation of salt and password) in a table:

Server[name, $s$, $h(s,$ password)] Alice → Server : "Alice", password

(4) The server has again the same table as in 1, but this time different messages are sent. First, Alice sends her name to the server. The server chooses a random number $n$ and sends it to Alice's local computer, subsequently Alice answers with the hash of the concatenation of $n$ and her password:

Server[name, password] Alice → Server : "Alice"
Server → Alice : $n$
Alice → Server : $h(n,$ password)

(5) The server has the same table as in 3, but this time different messages are sent. First, Alice sends her name to the server. The server chooses a random number $n$ and sends it to Alice's local computer together with the salt $s$, subsequently Alice answers with $h(n, h(s,$ password):

Server[name, $s$, $h(s,$ password)] Alice → Server : "Alice"
Server → Alise : $n$, $s$
Alice → Server : $h(n, h(s,$ password$))$

**The task**:

- For each protocol indicate how the server can authenticate Alice. In protocol 2, for example, the server can authenticate Alice by computing the hash of the password sent by Alice, and comparing it to the stored hash in the table.
- For the following pairs of protocols give an attack that will succeed for the first but fail (or not succeed very easily) for the second.
  - (1) and (2)
  - (1) and (3)
  - (3) and (4)
  - (3) and (5)

An example of an attack that will succeed for protocol 2 but not so easily for protocol 3 is a dictionary attack. An attacker 'steals' the table of the server. Because of property (O) the attacker cannot restore passwords based on the stored hashes. However, the attacker can try to discover passwords by hashing all words from a dictionary and comparing the results with the rows in the table. The attack is considerably more difficult for protocol 3 because the attacker would have to try all words from the dictionary again for each row in the table.

B (3 points) The following protocol is a somewhat simplified version of the debit protocol between a Chipknip card ($C$) and a terminal ($T$). Both the card and the terminal have a transaction counter ($n_C$ and $n_T$ respectively) which are both increased at every transaction. The protocol uses a diversified key $K_C$ which is different for each card $C$.

1. $C \rightarrow T$ : $\mathrm{Id}_C, n_C$
2. $T \rightarrow C$ : $\mathrm{amount}, K_C\{\mathrm{amount}, \mathrm{Id}_T, n_T, n_C\}$
   $n_C + +$;
   write log;
   if $\Big((\mathrm{MAC}) \wedge (\text{content OK}) \wedge (\ \mathrm{amount} \leq \mathrm{balance})\Big)$
   balance = balance - amount;
   else
   return exception;
3. $C \rightarrow T$ : $K_C\{??\}$,
   $n_T\ + +$;
   write log;
   if (decryption OK) Authorize Debit.

Where ?? should include yes/no, and possibly $\mathrm{Id}_T$ , amount, ...

Note that after step 2 the card knows that the terminal is authentic, since the terminal can generate $K_C$ . Yet, only after step 3 the terminal knows that the card is authentic.

**The task.** Complete and change the protocol so that mutual challenge-response based authentication is used instead of authentication based on transaction counters. Assume that both card and terminal have a proper random number generator. Explain your answer.

C (3 points) The encryption methods you have seen until now were all deterministic. This means: given a message $m$ and a key $k$, the outcome of the encryption $k\{m\}$ is always the same. In this exercise we look at some of the problems caused by this.

(a) Every Friday evening Alice sends a message to Bob "Will you go out with me tonight?" (in plaintext). Bob, who does not want Eve to be able to read the answer sends an encrypted message back: $k_{AB}\{\mathit{Yes}\}$ or $k_{AB}\{\mathit{No}\}$, where $k_{AB}$ is a secret key shared between Alice and Bob. Explain why this causes problems for both the secrecy and the integrity of Bob's answer.

(b) The problem of secrecy is solved in general by randomizing Bob's message. Instead of simply sending $k_{AB}\{\mathit{Yes}\}$ or $k_{AB}\{\mathit{No}\}$, Bob chooses a big random number (a nonce) $n$, and sends $k_{AB}\{\mathit{Yes}, n\}$ of $k_{AB}\{\mathit{No}, n\}$. Alice ignores $n$ and checks the real answer. Explain why this solves secrecy, but not integrity.

D (1 point) Mono-alphabetic substitution using the standard alphabet has a key space of size $26! \approx 4{\cdot}10^{26}$ (approximately 88 bits). Explain why it is easy to break anyway.