Security
Assignment 6, Friday, October 20, 2017

Deadline: Monday, November 13, 09:00 sharp!

Goals: After completing these exercises successfully you should be able to

- design protocols to achieve freshness goals;
- correctly identify properties of hash functions;
- reason about practical applications of hash functions.

Marks: You can score a total of 100 points.

1. **(0 points)** In yesterday's lecture, an app was demonstrated that reads and shows the data stored on Dutch identity documents: ReadID - NFC Passport Reader.
   (a) If you have (access to) an NFC-enabled Android, download the app and try reading your own identity document.
   (b) Find out whether or not the company behind ReadID, InnoValor, collects the data from the documents that are read via the app.

2. **(35 points)** In last week's lecture, we saw the following protocol to set up a symmetric key using a trusted third party that supplied 'tickets', containing a new key K_{AB} (i.e. the TTP generates a new K_{AB} for each execution of the protocol).

   1. $B \longrightarrow TTP$: "key request for B to A"
   2. $TTP \longrightarrow B$: $K_B\{K_{AB}\},\ ticket = K_A\{K_{AB}\}$
   3. $B \longrightarrow A$: $ticket, K_{AB}\{m\}$

   The lecture slides suggest several additional 'services' that the protocol could provide. In this exercise, we will extend the above protocol to better resist replay attacks. Incrementally extend the protocol such that:

   (a) .. Bob does not accept a replayed K_{AB};
   (b) .. Bob does not accept a ticket that does not belong to the K_{AB} he receives with it;
   (c) .. the TTP does not accept a replayed key request;
   (d) .. Alice does not accept a replayed ticket (and thus: a replayed encrypted message). *Hint:* involve Alice at an earlier point in the protocol.

3. **(35 points)** In this exercise we consider the following properties for hash functions $\mathcal{H}$:

   (P) Preimage resistance: given a bit string y output by $\mathcal{H}$, it is infeasible to find a bit string x such that $\mathcal{H}(x) = y$,
   (P2) Second preimage resistance: given a bit string x it is infeasible to find a different bit string x' $(x \neq x')$ such that $\mathcal{H}(x) = \mathcal{H}(x')$.
   (C) Collision resistance: it is infeasible to find two bit strings $x \neq x'$ s.t. $\mathcal{H}(x) = \mathcal{H}(x')$,
   (F) Fixed length: the length of output is fixed and does not depend on the input size (this is usually not a formal requirement, but it is, in practice, often the case).

   We define possible hash function candidates h. Do they meet each of the properties (P), (P2), (C) and (F)? Explain briefly for all properties (for each hash function).

   (a) The function is defined as $h(x) := 11111011100$.

(b) The function is defined as $h(x) := x \| x$ (where $\|$ is concatenation, that is, x is repeated).

(c) Assume that $\mathcal{H}_1, \mathcal{H}_2$ are two hash functions that meet requirements (P), (P2), and (F). The function $\mathcal{H}_1$ also meets (C), whereas $\mathcal{H}_2$ does not meet (C). The function is defined as $h(x) := \mathcal{H}_1(x) \| \mathcal{H}_2(x)$.

(d) Assume that $\mathcal{H}$ is a hash function that meets all four requirements (C), (P), (P2), and (F). The function is defined as $h(x) := \mathcal{H}(|x|)$ (where $|x|$ is the bit length of x).

(e) Assume that $\mathcal{H}$ is a hash function with output bit-length N that meets all four requirements (C), (P), (P2), and (F). The candidate function is defined as

$$h(x) := \begin{cases} 0^N, & \text{if } x = 0^{|x|} \\ \mathcal{H}(x), & \text{otherwise;} \end{cases}$$

that is, if the input of function h is such a bit-string all bits are zero, it outputs an all-zero bit-string of length N. Otherwise, it outputs the same as $\mathcal{H}$.

4. **(30 points)** The following authentication protocol makes use of the Lamport hash construction (sometimes simply called 'hash chain'). The construction simply consists of repeatedly applying a hash function to an input value. We write $h^n(x)$ to denote hashing x n times, *e.g.* $h^3(x) = h(h(h(x)))$. Each user chooses a password pw and hashes this n times. He/she then sends it to the server. Let us assume that initially, $n = 10\,000$. The server then stores a tuple (user, n, $Y = h^n(pw)$) for each user in a database. Users can now authenticate to the server using the following protocol:

$$\begin{array}{lcll} 1. & A \longrightarrow S & : & A \\ 2. & S \longrightarrow A & : & n \\ 3. & A \longrightarrow S & : & X = h^{n-1}(pw) \end{array}$$

The server checks if $h(X) = Y$, then decrements n and sets $Y := X$. So, after a successful run of this protocol the server holds a new tuple (user, $n-1$, $Y = h^{n-1}(pw)$).

(a) Can you think of an attack (here, relay or simple man-in-the-middle is not considered an attack), after which the adversary can gain access multiple times without the user being present? Use the arrow notation ($A \longrightarrow B$: message) in your explanation.

(b) At some point $n = 0$. Is it safe to start over again and put n back to 10 000? Briefly explain your answer.

(c) The amount of hashes the user has to compute differs depending on n. Suppose the user locally stores

$$h^i(pw) \text{ for } i = 0, 1000, 2000, \ldots, 9000\,.$$

How many hash function evaluations does the user have to make on average (for a single evaluation of the protocol)?[1]

(d) One way to construct a one-time pad that uses a Lamport hash is by starting from the hash of a random starting value r and generating an infinite sequence of $h(r), h(h(r)), \ldots$. Is this way to construct a one-time pad secure? Briefly motivate why, or describe an attack. *Hint:* consider what happens when a part of the plaintext is predictable.

[1] This approach is a naive solution to the pebbling problem. Schoenmakers recently introduced an optimal algorithm: `http://www.win.tue.nl/~berry/pebbling/Schoenmakers-FC16-slides.pdf`.