Security

Assignment 7, Friday, October 21, 2016

Handing in your answers: For the full story, see

`http://www.sos.cs.ru.nl/applications/courses/security2016/exercises.html`

To summarize:

- Include your name and student number **in** the document (they will be printed!), as well as the name of your teaching assistant (Hans or Joost). When working together, include **both** your names and student numbers.
- Submit one single **pdf** file – when working together, only hand in **once**.
- Hand in via Blackboard, before the deadline.

Deadline: Monday, November 14, 09:00 sharp!

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

- reason about practical applications of hash functions.

Marks: You can score a total of 100 points.

1. **(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:

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

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) 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.

   (d) Another way to construct a one-time pad using a Lamport hash is to 'reverse' the above approach, to obtain $h^{1000}(r), h^{999}(r), h^{998}(r), \ldots, h(r)$. (The length of the pad in this case is 1000 times the output length of the hash function, after which a new random value is chosen for a new pad.) What is the security problem with this construction?

2. **(35 points)** Small embedded devices typically do not have a lot of space available to do complex computations. More importantly, they should be cheap to produce. Still they need to be able to communicate securely. In this exercise, we imagine a device that can only compute a hash function (and basic operations such as XOR's), but still wants to achieve confidentiality and integrity. Assume we share a key k with the device, and we have a hash function $\mathcal{H}$ that outputs 256-bit hashes.

   Assume all messages are 512 bits. To encrypt, we first split a message in two parts of 256 bits: $m = m_1 \| m_2$ (as always, $\|$ is concatenation). We compute the encrypted message $k\{m\} = c_1 \| c_2$ in two parts, as follows:

$$c_1 = \mathcal{H}(k \| m_2) \oplus m_1 \qquad \text{and} \qquad c_2 = m_2 \oplus \mathcal{H}(c_1 \| k).$$

   (a) Assuming you have the key k, how do you obtain m from $k\{m\} = c_1 \| c_2$? Show what computations need to be done.

   To guarantee integrity, we compute a MAC over the ciphertext. As we only have a hash function, we use a variant of the so-called HMAC construction[1]: we compute two tags $t_1 = \mathcal{H}(k \| c_1)$ and $t_2 = \mathcal{H}(k \| c_2)$.

   (b) Explain what property of the hash function[2] is most important for this HMAC construction to prevent outsiders from creating valid tag-ciphertext pairs (i.e. some (t', c')) without having the key.

   Assume that later, by coincidence, we send a message m' for which it holds that $m'_2 = c_2$, for some c_2 from an earlier message.

   (c) Somehow an adversary finds out that this has happened. Of course, they have been recording everything that has ever been sent. What other (part of a) message can they now recover?

   (d) Is this still a problem if we use only one tag $t = \mathcal{H}(k \| c_1 \| c_2)$ instead of a separate t_1 and t_2? Why (not)?

3. **(35 points)** Hash functions play an important role in the BitTorrent protocol. When using BitTorrent, users receive pieces of a large file from different providers (typically referred to as 'seeders'). When a user wants to obtain a certain file, he needs a '.torrent' file. Amongst other things, this file can contain a list of hashes. These hashes are used to guarantee the integrity of the individual pieces, allowing a user to check each piece when he receives it.

   (a) An attacker tries to replace one piece of the original piece with a new one. What property of the hash function is important here so that the attack is detectable?

   (b) Say Alice wants to download a file of 2 GB (*e.g.* an authentic Linux distribution), and it is split in pieces of 16 KB each. Suppose that the .torrent file she uses contains a list of SHA-1 hashes. What is the minimum size of that .torrent file? (*Hint:* A SHA-1 hash is 160 bits = 20 bytes)

   As (b) shows, that can be quite a large file to deal with. We could resolve this by increasing the size of each piece so that we do not need as many hashes. That would however make it more difficult for peers to quickly share their pieces. An alternative approach is to use *binary hash trees.*

   Consider Figure 1. On the circles along the bottom (the *leaf nodes*), we place the hash values of the different pieces: $N^0_0 = h(\text{piece}_0), N^0_1 = h(\text{piece}_1), N^0_2 = h(\text{piece}_2), \ldots$. Each of the higher nodes in the tree contains the combined hash of its children. For example, the

[1]Note that we use a simplified construction here: it is not secure when used with certain popular hash functions.
[2]Preimage resistance (P), 2^{nd} preimage resistance (P2) or collision resistance (C)

value in the first node of the next layer would be $N_0^1 = h(N_0^0 \| N_1^0) = h(h(\text{piece}_0) \| h(\text{piece}_1))$. The node above that would be $N_0^2 = h(N_0^1 \| N_1^1)$, *etc.* We continue doing this all the way to the top of the tree. The top of the tree is called the root R, which is always contained in the .torrent file.

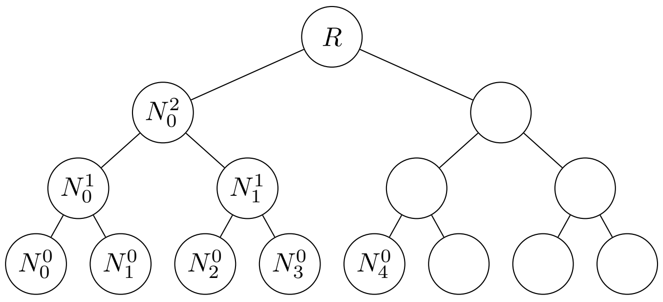

Figure 1: binary hash tree

(c) Suppose the .torrent file does not contain a list of all hashes, but only the root hash. At which stage would you be able to check the integrity of the pieces you receive from others? Why?

Instead of just sending you the piece, a seeder should now also send you a few of the nodes in the tree. This guarantees that you have enough information to calculate the root node, using his piece and the hashes he sends along. You can then compare it to the root node in the '.torrent' file. If it matches, the piece was authentic!

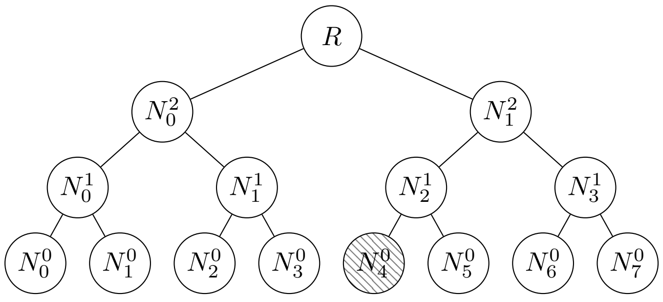

Figure 2: binary hash tree with a marked leaf node

(d) Suppose someone sends you piece$_4$. Now you can compute $N_4^0 = h(\text{piece}_4)$, marked gray in Figure 2. Looking at the figure, what other N-values (nodes) would he need to send you before you can compute R, to check if the piece was correct?

(e) Suppose we have a file that consists of 1024 pieces. How many hashes would need to be sent along with each of the pieces?

This exercise was only a very slight abstraction from reality. Have a look at `http://bittorrent.org/beps/bep_0030.html` if you are interested in what this looks like in practice.