SECURITY

Assignment 6, Saturday, October 21, 2017

S1013793 Carlo Jessurun
S1013792 Tony Lopar
Radboud University

Teaching assistant: Joost Rijneveld

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

Solutions 2

A. Adding a nonce to the second line.

TTP → B: $K_B\{K_{AB}, N_B\}$, ticket $K_A\{K_{AB}\}$

B. In order to check the key in the ticket, Bob should be able to see the content of the ticket. Therefore he should know K_A in order to decrypt the ticket, so he can verify it.

TTP → B: $K_B\{K_{AB}, K_A, N_B\}$, ticket $K_A\{K_{AB}\}$

C. We can do this by adding a time based nonce to the request.

B → TTP : "key request for B to A", N_B

D. TTP could ask Alice for a nonce and put this in the ticket. Alice may verify whether this nonce is in the ticket that she receives from Bob. If she receives the same nonce two times, she would only accept it the first time.

B → TTP : "key request for B to A", N_B
TTP → A: "GIMME YOUR NONCE GIRL"
A → TTP: N_A
TTP → B: $K_B\{K_{AB}, N_B, N_A\}$, ticket $K_A\{K_{AB}, N_B, N_A\}$
B → A : ticket,$K_{AB}\{m\}$

Assignment 3

3. (35 points) In this exercise we consider the following properties for hash functions H:
(P) Preimage resistance: given a bit string y output by H, it is infeasible to find a bit string x such that H(x) = y,
(P2) Second preimage resistance: given a bit string x it is infeasible to find a different bit string x′ (x ≠ x′) such that H(x) = H(x′).
(C) Collision resistance: it is infeasible to find two bit strings x ≠ x′ s.t. H(x) = 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 H1 , H2 are two hash functions that meet requirements (P), (P2), and (F). The function H1 also meets (C), whereas H2 does not meet (C). The function is defined as h(x) := H1(x)||H2(x).
(d) Assume that H is a hash function that meets all four requirements (C), (P), (P2), and (F). The function is defined as h(x) := H(|x|) (where |x| is the bit length of x).
(e) Assume that 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 H.

Solutions 3

Given the proposed hash function in a we got the following result for the four properties consisting of preimage resistance, secondary preimage resistance, collision resistance and fixed length. We will repeat this table for every proposed hash function below.

A. *The function is defined as h(x) := 11111011100.*

Prop	Status	Reason
P	**FALSE**	The function is not pre-image resistant, since for every input the hash will generate the same output which means we will get the same y for every x.
P2	**FALSE**	The hash function doesn't meets this property, since a different value will result exactly the same hash, so we may find m' that breaks this requirement by changing one character.
C	**FALSE**	The hash doesn't meet this requirement, because there is a collision between all outputs of the hash.
F	TRUE	The hash meets this requirement, since all hashes will have the length of 11 characters.

B. *The function is defined as h(x) := x // x (where // is concatenation, that is, x is repeated).*

Prop	Status	Reason
P	**FALSE**	The function doesn't meet this requirements. Since x is repeated in the function, we can simply find x by removing the duplication from y.
P2	**FALSE**	The function does meet this requirement, since for x a different repetition will result as y.
C	**TRUE**	The function does meet this requirement, since for different x, we will get different repetitions.
F	**FALSE**	The function doesn't meet this requirement, since the length of y is dependent on the size of x.

C. The function: h(x) := H1(x) // H2(x).

Prop	Status	Reason
P	**TRUE**	Since H1 and H2 meet this requirement, we cannot find the initial message from the concatenation of both hashes.
P2	**TRUE**	Since the concatenation exists from both parts which meet the requirement. The output of this function will also meet it.
C	**TRUE**	The function meets this requirement, because H1(x) meets this requirement. This means that for two different messages the first part of the hash will never cause a collision. Since the first part already never can cause a collision, it's irrelevant whether the second part of the hash is collision resistance
F	**TRUE**	The function meets this requirements, since H1 and H2 both have a fixed length. This means that a concatenation of both will also have a fixed length.

Ik denk dat de uitwerking hierboven correct is. Echter twijfel ik aan h(x) := H1(x)||H2(x).. Daarnaast, lees je de vergelijking als dat h(x) het product is van de berekening H1(x)||H2(x)?

D. *"Assume that H is a hash function that meets all four requirements."*
h(x) := H(|x|) (where |x| is the bit length of x).

Prop	Status	Reason
P	**TRUE**	This requirement holds, because from the hash we may find the length of m but to find exactly that specific m from the hash will be as good as impossible.
P2	**FALSE**	This requirement does not hold, because for two different messages with the same length, we will have the same output of the hash.
C	**FALSE**	This requirement does not hold, because we can create a collision by taking two random different message of the same length. These messages will be hashed to the same hash.
F	**TRUE**	This requirement holds, since it holds for H which means it doesn't depend on the length of the input.

E. Assume that H is a hash function with output bit-length N that meets all four requirements (C), (P), (P2), and (F).

Prop	Status	Reason
P	**FALSE**	This requirement doesn't hold, because if we have an hash only containing zero's, then we know that the original message only contained zero-bits.
P2	**TRUE**	This requirement holds, since only one value is explicitly specified. This means we cannot find a different message with the same hash, since we already know H(x) meets this requirement.
C	**TRUE**	The function meets this property, because it holds for H(x). Since only one value is explicitly specified, this holds for all values.
F	**TRUE**	This requirement holds, because the h(x) will generate an output of the same length for both cases.

Assignment 4

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:

1. $A \rightarrow S : A$
2. $S \rightarrow A : n$
3. $A \rightarrow S : X = h^{n-1}(pw)$

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 → 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

$h^i(pw)$ for i = 0,1000,2000,...,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)), ... 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.

Solutions 4

A. The only attack we might imagine is a session hijack, since man-in-the-middle and relays are not considered attacks here. By stealing the session of a user, we might then observe the state of that connection. Given that we could observe the state of n, we could then reuse that session with that given n. The downside to doing this attack is that if the user decides to re authenticate, there would be a mismatch and a new session would have to be started (this should ring some alarm bells).
B. No, because we may use the earlier intercepted messages to login. When we eavesdropped all messages from 10.000 to 0, then we may repeat a message to login in this case. Here it would be better to use a nonce or time based token as n.
C. 10, since there are ten different options possible for i.
D. Repetition in key of one time pad, may reveal multiple messages when part of a message is known B
E. being able to predict a part of the plaintext can help us to discover the associated part of the hash. From this we may recover r. When we recovered r, then we may find out the whole plaintext, since the whole stream depends on r. If we have found the possible message we can verify it by comparing the hash with the hash of the possible message.