Dr. Michael Reyer

Tutorial 7
- Proposed Solution -

Friday, December 7, 2018

Solution of Problem 1

a) In order to break Lamport's protocol we need to compute the $(A, i+1, w_{i+1})$ given (A, i, w_i) from the previous transmission i. Since the computation of A and $i+1$ is trivial, we only need to compute the following inverse hash function:

$$w_{i+1} = H^{t-i-1}(w) = H^{-1}(H^{t-i}(w)) = H^{-1}(w_i).$$

If H is a *secret* one-way function, this step is clearly infeasible. However, even for a *public* one-way function, this step is also infeasible, since the computing w_{i+1} and H^{-1} is infeasible given H and w. Hence, using a secret function is not required.

b) Check if each of the four basic requirements on hash functions is necessary:

1. H is easy to compute:
   Recall: *Given $m \in \mathcal{M}$, $H(m)$ is easy to compute.*
   This not required, but still a very useful property to provide an efficient protocol.
2. H is preimage resistant: (required ✓)
   Recall: *Given $y \in \mathcal{Y}$, it is infeasible to find m, such that $H(m) = y$.*
   Otherwise, $w_i = H(w_{i+1})$ could be broken, see a).
3. H is second preimage resistant: (required ✓)
   Recall: *Given $m \in \mathcal{M}$, it is infeasible to find $m' \neq m$, such that $H(m) = H(m')$.*
   Otherwise, the attacker would be able to find a w' such that $H(w') = H(w_{i+1})$.
4. H is collision-free:
   Recall: *It is infeasible to find $m \neq m' \in \mathcal{M}$ with $H(m) = H(m')$.*
   Although finding an arbitrary collision would indeed break the system, it will affect a random chain of passwords in this scheme with negligible probability.

c) The discrete logarithm problem is hard to solve in $\mathbb{Z}_p^*$:
It is hard to determine x in $a^x \equiv y \mod p$ for given values of the primitive element a modulo p and y.
Lamport's protocol in terms of the discrete logarithm problem is described by:

- Functions and Parameters:
  Use the one-way function $f : \mathbb{Z}_p \to \mathbb{Z}_p^*$ with $w \to a^w \mod p$.
  Choose a secret value $w \in \{2, ..., p-2\}$ and a primitive element $a \mod p$.
  Choose t, the maximal number of identifications.
  Select the initial value $w_0 = f^t(w)$.

- Protocol steps:
  Compute next session key $f^{t-i}(w) = w_i$.
  Session authentication $A \to B : (A, i, w_i)$.
  B checks if $i = i_A$ and $w_{i-1} = a^{w_i} \mod p$ is true.
  If correct, B accepts, sets $i_A \leftarrow i_A + 1$ and stores w_i for the next sesssion.

d) *Man-in-the-middle attack* on Lamport's protocol:
Let E intercept the current key w_i from A. E uses it for authentication as A at B. Furthermore, if E gains access to the initial value w and knows the current session number i, the protocol is completely broken.

Solution of Problem 2

a) Claimant Alice (A) wants to prove her identity to verifier Bob (B). This identification is done for a fixed password by comparing its hash value to a stored hash value. The password pwd is sent without protection. B calculates $h(pwd)$ and compares it with the stored hash value, to verify the identity of A.

In a *replay attack*, eavesdropper Eve (E) intercepts the password and impersonates A by reusing the password in a later session:

$A \to E :$ pwd (plain password transmission interecpted by E)

$E \to B :$ pwd (impersonating A)

By encrypting the password E still may impersonate A. However, E will not know the password pwd.

b) Consider the following authentication protocol:

1) $A \to B : r_A$ (A challenges B)
2) $B \to A : E_K(r_A, r_B)$ (B responds to A and challenges A)
3) $A \to B : r_B$ (A responds to B)

In the *reflection attack*, E uses A to reveal the correct responds:

$A \to E : r_A$ (challenge)

$E \to A : r_A$ (the same challenge back)

$A \to E : E_K(r_A, r_{A'})$ (response)

$E \to A : E_K(r_A, r_{A'})$ (the same response back)

$A \to E : r_{A'}$ (second response)

$E \to A : r_{A'}$ (the same second response back)

Remark: No user B is involved here, only the 'reflection' of A.

Such an attack can be easily avoided by checking, if a challenge has been used already. Then it obviously cannot be reused. If in step 3) A sends $E_K(r_B)$ or $h(r_B)$ then a reflection attack is not prevented, but r_B, which might be used as joint secret key, is not known by E.

c) Consider the following mutual authentication protocol:

1) $A \to B : r_A$ (challenge)
2) $B \to A : r_B, S_B(r_B, r_A, A)$ (response and 2nd challenge)
3) $A \to B : r'_A, S_A(r'_A, r_B, B)$ (2nd response)

The *interleaving attack* uses the information of simultaneous sessions:

$E \to B : r_A$ (1st session 1))
$B \to E : r_B, S_B(r_B, r_A, A)$ (1st session 2))
$E \to A : r_B$ (2nd session 1))
$A \to E : r'_A, S_A(r'_A, r_B, B)$ (2nd session 2))
$E \to B : r'_A, S_A(r'_A, r_B, B)$ (1st session 3))

Now E can impersonate as A to B. Remark: In this case the sessions of two protocols are interleaved (overlapped) like in a man-in-the-middle attack. This attack can be avoided by exchanging 2) by

2') $B \to A : h(r_B), S_B(E_A(r_B), r_A, A)$.

E_A is an encryption with A's public key, i.e., A might calculate r_B and check, if $h(r_B)$ is correct. The roles of h and E_A might be exchanged.

Solution of Problem 3

Useful sources to study the Kerberos protocol are, e.g.:

- *Trappe, Washington - Introduction to Cryptography with Coding theory*
- *http://en.wikipedia.org/wiki/Kerberos_(protocol)*

Unilateral authentication by the Kerberos protocol with a ticket granting server:

(1) *User logon, A requests client authentication at T to use G:*
$A \to T : A, G$

(2) *T grants client authentication for A at G:*
T generates session key k_{AG}.
T generates a ticket granting ticket (TGT): $TGT = G, E_{k_{TG}}(A, t_1, l_1, k_{AG})$.
$T \to A : E_{k_{AT}}(k_{AG}), TGT$

(3) *A requests client authentication for service at G:*
A recovers k_{AG} using the shared key k_{AT}.
A generates an authenticator $a_{AG} = E_{k_{AG}}(A, t_2)$.
$A \to G : a_{AG}, TGT, B$

(4) *G grants service to A:*
G recovers A, t_1, l_1, k_{AG} from the TGT using k_{TG}.
G recovers A, t_2 from a_{AG} using k_{AG}.
G checks if the time stamp is within the validity period $(t_2 - t_1) < l_1$.
G verifies A if authenticator and the ticket are correct.
G generates session key k_{AB} and service ticket ST using k_{BG}: $ST = E_{k_{BG}}(A, t_3, l_2, k_{AB})$.
$G \rightarrow A : ST, E_{k_{AG}}(k_{AB})$

(5) *A communicates with B with the authenticated service of G:*
A recovers k_{AB} using k_{AG}.
A generates authenticator $a_{AB} = E_{k_{AB}}(A, t_4)$.
$A \rightarrow B : a_{AB}, ST$
B recovers A, t_3, l_2, k_{AB} from ST using k_{BG}.
B recovers A and t_4 from a_{AB} using k_{AB}.
B checks if the time stamp is within the validity period $(t_4 - t_3) < l_2$.
B verifies A if authenticator and service ticket are correct.
Then, A is successfully authenticated to B.

(6) **Optional extension to mutual authentication**
$B \rightarrow A : E_{k_{AB}}(t_4)$