Homework 3 Solutions

CS161 Computer Security, Spring 2008

1. **Authentication and Key Exchange**

   The Needham-Schroeder protocol has a number of variants; in particular, the version in Figure 12.3 of the Gollman textbook differs from the one presented in lecture. As explained in Section 12.3.3, the textbook variant is vulnerable to a replay attack which tricks B into reusing an old session key (which may have since been compromised).

   (a) (3 points)

   Modify this version of the Needham-Schroeder protocol so that it prevents this attack. You may assume that all three parties have synchronized clocks, but you must not add any messages to the protocol. That is, the protocol should still consist of a message from A to S, S to A, A to B, B to A, and A to B, but the fields in each message may differ from the original protocol. Express your answer in the protocol notation of Figure 12.3 and note any checks conducted by each party after receiving each message.

   Answer: The server may add a timestamp t as shown below.

   $$
   \begin{aligned}
   &A \rightarrow S : A, B, n_a \\
   &S \rightarrow A : E_{K_{as}}(n_a, B, K_{ab}, E_{K_{bs}}(K_{ab}, A, t)) \\
   &A \rightarrow B : E_{K_{bs}}(K_{ab}, A, t) \\
   &B \rightarrow A : E_{K_{ab}}(n_b) \\
   &A \rightarrow B : E_{K_{ab}}(n_b - 1)
   \end{aligned}
   $$

   After receiving the message $E_{K_{bs}}(K_{ab}, A, t)$, B should check (in addition to the checks in the original protocol) that t is within some small window around the current time and abort the protocol if not.

(b) (5 points)

If synchronized clocks are not available, it is also possible to prevent this replay attack by adding additional messages to the protocol and altering the content of other messages. An incomplete version of such a protocol is given on the following page.

$$
\begin{aligned}
&A \rightarrow B : A \\
&B \rightarrow A : \text{\_\_\_\_\_} \\
&A \rightarrow S : A, B, n_a, \text{\_\_\_\_\_} \\
&S \rightarrow A : E_{K_{as}}(n_a, B, K_{ab}, \text{\_\_\_\_\_}) \\
&A \rightarrow B : \text{\_\_\_\_\_} \\
&B \rightarrow A : E_{K_{ab}}(n_b) \\
&A \rightarrow B : E_{K_{ab}}(n_b - 1)
\end{aligned}
$$

In this version, A and B exchange an initial round of messages before A contacts S. After that point, the rest of the protocol is very similar to the version of Needham-Schroeder given in Figure 12.3; in particular, the last two messages are the same.

Each of the blanks represents one or more fields that have been omitted. The size of the blanks is not meant to indicate anything about the content of those fields.

Give values for these omitted fields so that the resulting protocol is not vulnerable to the replay attack. Try to avoid including anything beyond what is necessary.

Answer: The purpose of the initial exchange between A and B is to give B an opportunity to provide a nonce which will demonstrate the freshness of the key it receives from S. One way to do so is with the fields given below.

Blank 1: $E_{K_{bs}}(n'_b)$
Blank 2: $E_{K_{bs}}(n'_b)$
Blank 3: $E_{K_{bs}}(K_{ab}, n'_b, A)$
Blank 4: $E_{K_{bs}}(K_{ab}, n'_b, A)$

2. **More Authentication and Key Exchange**

Consider the following protocol, which is an alternative to Needham-Schroeder.

$$A \rightarrow B : n_a, A, B, E_{K_{as}}(n'_a, n_a, A, B)$$
$$B \rightarrow S : n_a, A, B, E_{K_{as}}(n'_a, n_a, A, B), E_{K_{bs}}(n_a, A, B, n_b)$$
$$S \rightarrow B : n_a, E_{K_{as}}(n'_a, K_{ab}), E_{K_{bs}}(n_b, K_{ab})$$
$$B \rightarrow A : n_a, E_{K_{as}}(n'_a, K_{ab})$$

In the above, n_a and n'_a are nonces selected by A and n_b is a nonce selected by B.

(a) (2 points)

Do you think this protocol is vulnerable to the same attack as the Needham-Schroeder protocol? That is, is it possible to replay messages in order to cause A or B to reuse an old session key? Briefly explain why you think it is or is not vulnerable to such an attack.

Answer: No, it is not vulnerable to the same attack. The message from which A obtains the session key ($E_{K_{as}}(n'_a, K_{ab})$) contains a nonce n'_a from A and will be rejected if A did not just select that value. Likewise, the message $E_{K_{bs}}(n_b, K_{ab})$ contains the nonce n_b from B and will be rejected if it is not fresh.

(b) (7 points)

In any case, that protocol is vulnerable to a less severe attack. Specifically, it is possible for an adversary to trick A and B into establishing a session with keys that do not match. That is, A will subsequently attempt to communicate with B using some session key K_{ab} while B will being using some key $K'_{ab} \neq K_{ab}$.

Show how this can be done. Assume the attacker can intercept messages and alter, drop, or replay them, but assume that the attacker has not compromised any session keys or any of the long term keys A and B share with S.

Answer: The attacker waits until the first three messages have been received, at which point B will have received the session key K_{ab} and will send the fourth message. The attacker intercepts and drops that fourth message, then replays the second message $(A, B, E_{K_{as}}(n'_a, n_a, A, B), E_{K_{bs}}(n_a, A, B, n_b))$ to the server. This causes the server to generate a new session key K'_{ab} and reply to the attacker with $n_a, E_{K_{as}}(n'_a, K'_{ab}), E_{K_{bs}}(n_b, K'_{ab})$. The attacker may now send $n_a, E_{K_{as}}(n'_a, K'_{ab})$ to A, causing A and B to carry on from this point using different session keys.

3. **Generating Random Values**

In order to generate a cryptographic key or a seed for a pseudorandom number generator, it is necessary to collect some (truly) random data. Common sources of random data on a system include a high resolution (e.g., nanosecond) clock, latencies between disk seeks, delays between keystrokes or mouse movements, etc.

All of these sources contain some amount of truly unpredictable information, but they are in general not sources of uniform, uncorrelated bits. In order to obtain uniformly distributed random values from such sources, the common practice is to pass them through a cryptographic hash function. However, it is possible to condition random data through simpler methods.

Suppose you wish to generate some number of bits, each of which should be 0 or 1 with equal probability, independent of the other bits. You have a biased coin which turns up tails (0) with some unknown probability p, where $0 < p < 1$, and turns up heads (1) with probability $1-p$. However, you do not have a computer with which to compute a hash function. In this scenario, the following simple algorithm can be used.

1. Flip the coin twice.
2. If the outcome is tails, then heads, write down a 0. If the outcome is heads, then tails, write down a 1. Otherwise, do not write anything.
3. Repeat from Step 1 until the required number of bits has been generated.

If we view the coin flips as generating a string of input bits, this algorithm can be viewed as passing over the string two bits at a time, applying the following transformation.

$$\begin{aligned} \text{“00”} &\rightarrow \text{“ ”} \\ \text{“01”} &\rightarrow \text{“0”} \\ \text{“10”} &\rightarrow \text{“1”} \\ \text{“11”} &\rightarrow \text{“ ”} \end{aligned}$$

For example, the input string

“10 00 00 01 01 10 01 10 00 00 01 00 10”

would be translated to

"1 0 0 1 0 1 0 1".

Regardless of the value of p, each output bit is equally likely to be 0 or 1 because the pair of input bits "01" has the same probability of occurring as the pair "10". Furthermore, the probability of each output bit being 0 or 1 will be independent of the other output bits.

(a) (3 points)

Using this method, if the number of input bits is ℓ (which may be assumed to be even) and the probability of each input bit being 0 is p, what is the expected number of output bits generated?

Answer: $\ell p(1-p)$

(b) (8 points)

Design a more efficient scheme, that is, one which provides a higher expected ratio of output bits to input bits.

Like the one above, your scheme should take the form of a mapping from groups of input bits to groups of output bits, and you should specify it in this form.

Each of the output bits produced by your scheme should be 0 or 1 with equal probability, independent of the other output bits. For any value of p such that $0 < p < 1$, your scheme should produce more output bits on average than the given scheme.

Answer: One way to devise an improved scheme is to take larger groups of input bits at a time. Note that any set of input strings which each have the same number of 0's and 1's will all occur with equal probability. So an easy way to ensure that the output bits will be uniform and independent is to group the input strings by how many 1's and 0's they have, and assign output strings accordingly.

The following page gives the mapping for a scheme designed in this way. The rows have been rearranged to make the idea behind the scheme more clear.

"0001" → "00"
"0010" → "01"
"0100" → "10"
"1000" → "11"
"0111" → "00"
"1011" → "01"
"1101" → "10"
"1110" → "11"
"0011" → "00"
"0101" → "01"
"0110" → "10"
"1001" → "11"
"1010" → "0"
"1100" → "1"
"0000" → ""
"1111" → ""

To see why this works, note that the first four input strings listed each have exactly one 1 and thus occur with equal probability. These are mapped to "00", "01", "10", and "11". Likewise, the four input strings after those occur with equal probability and have the same mapping. The four input strings listed after those each have two 0's and two 1's and again are given the same mapping. The remaining two possible input strings with two 0's and two 1's are mapped to "0" and "1", and, finally, the all 0's and all 1's strings are mapped to the zero length string.

(c) (4 points)

If the number of inputs bits is ℓ and the probability of each bit being 0 is p, what is the expected number of bits generated by your new scheme?

Answer: The expected number of bits generated from a single group of four input bits is

$$4 \cdot 2p^3(1-p) + 4 \cdot 2p(1-p)^3 + 4 \cdot 2p^2(1-p)^2 + 2p^2(1-p)^2$$
$$= -6p^4 + 12p^3 - 14p^2 + 8p$$

So the overall expected number of bits is

$$\left\lfloor \frac{\ell}{4} \right\rfloor \left(-6p^4 + 12p^3 - 14p^2 + 8p\right) \ .$$