C2107 Tutorial (Data Authentication)

School of Computing, NUS

February 14, 2021

Remark: Tutorial 2 Question 5 to be covered in this week tutorial.

Solution

1. Yes. According to RFC 4086 recommendation (lecture note), 30 bits is sufficient to be secure against online attack. There are a total of $(26 \times 2 + 10 = 62)$ alphanumeric symbols. So each character in the password contribute to at least 5.9 bits. 10 characters would be 59 bits and hence more than sufficient. (To be rigorous, note that total number of passwords is 62^{10}. Assuming each password is equally likely, the entropy would be $\log_2 62^{10} > 10 \times 5.9 = 59$.)

2. No. Now, offline attack is possible. According to the assumption of attack scenario, attacker can first sniff and get h, r. Next, the attacker exhaustively search for the password p s.t. $h = SHA3(r\|p)$ without interacting with the system. The p that meets the equality test must be the correct password. The entropy of the password is $\log_2 62^{10} < 5.96 \times 6 = 59.6$. According to NIST recommendation, at least 128 bits are required. It is also short from the RFC 4086 recommendation of 60 bits.

3. Additional Remarks. The attack scenarios consider a passive eavesdropper who sniff the h and r, there are also scenarios where an attacker pretend to be the server and get the h and r.

4. Additional Remarks. You may have heard that WPA2 is "secure". That is not the full story. There are variants of WPA2. Try to find out the difference between LEAP and PEAP. There are variants of WPA2 that uses LEAP or PEAP (LEAP is vulnerable to offline attack and PEAP is secure against offline attacks). Fortunately, NUS uses PEAP.

1. (*Birthday attack*) The are about one hundred thousand hair glands on the scalp, and different persons have different number of hair glands. Suppose there are 1000 undergraduates in SoC. Is the probability high (i.e. more than 0.5) that there exists two SoC students having the same number of hair glands?

Solution

Let $M = 1,000$, and $T = 100,000$. We can see that the two numbers satisfy the following condition given in the lecture: $M > 1.17 \cdot \sqrt{T}$. Hence, the probability that there exist two students with the same number of hair glands is greater than 0.5.
(You can also apply the formula given in the lecture to calculate the actual probability, which is: $1 - e^{-M^2/2T} = 1 - e^{-5} \approx 0.99326$.)

2. (*Birthday attack*) Suppose the length of IV is 64, and during each encryption, the IV is randomly and uniformly chosen. In a set of 2^{33} ciphertext, is the probability that there are two ciphertext with the same IV greater than 0.5?

Solution

Let $M = 2^{33}$, and $T = 2^{64}$. Apply the condition to show that the probability is greater than 0.5.

3. (*Pitfalls: using hash as truly random number generator*) Cryptographic hash functions, for e.g. `SHA1` (recall that SHA1 produces 160-bit digests), are often employed to generate "pseudo-random" numbers. Given a short binary string s (this is also known as the *seed*), we can generate a pseudo-random sequence $x_1, x_2, x_3, \ldots$, where each x_i's is a 160-bit string in the following way.

   - Let $x_1 = \texttt{SHA1}(s)$, and $x_{i+1} = \texttt{SHA1}(x_i)$ for $i \geq 1$.

   Complex protocols usually contain many components. Here is one typical component: when given a message m, the following is carried out.

   (a) Server randomly chooses a 128-bit k and another 128-bit v.
   (b) The server encrypts m using AES CBC-mode, using k as the encryption key, and v as the IV.
   (c) The server broadcasts the ciphertext over the air, and keeps k for further usage.

   Bob implemented the above component. To choose the k and v in step (a), Bob first set the seed s to be a string of 160 zeros, and then obtained x_1 and x_2 as described above. Bob subsequently took the leading 128 bits of x_1 as the v; and the leading 128 bits of x_2 as k. Bob claimed the following:

   "Since `SHA-1` *produces a random sequence, the 128-bit key and the 128-bit IV are therefore random, thus meeting the specified security requirement."*

Assume, as usual, that an eavesdropper could obtain the ciphertexts, and that the mechanism used by Bob to generate the key is publicly known. Give a ciphertext-only attack that finds the key k, and explain why Bob's argument is wrong.

Solution

To generate the IV and key, Bob insecurely employed SHA-1 by giving a fixed string of 160 zeros as the seed s. Notice that SHA hash function family is a "pseudo" random number generator, and is "deterministic". Hence, the attacker can simply derive the key by repeating Bob's key generation process, i.e. by taking the leading 128 bits of `SHA-1`(`SHA-1`($000\ldots000$)).

4. (*Still insecure. Using non-cryptographic secure pseudo random number generator*)

   Consider the same scenario in question 3. Bob realised his mistake and changed his program. The updated program chose the s by using the following (see Lecture 1 Cryptographic Pitfalls).

```
#include <time.h>
#include <stdlib.h>
      srand(time(NULL));
      int s = rand();
```

   After the seed s was chosen, following the same step in question 3, Bob applied `SHA1`, generated x_1, x_2 and extract the 128-bit v and k.

   If you are not familiar with C, the above can be replaced by `java.util.Random` as mentioned in Lecture 1.

   Explain why the above is not secure by giving a ciphertext-only attack that obtain the AES key. As usual, we assume Kerckhoffs's principle (i.e. a strong adversary knows the algorithm and all other information except the secret key.)

Solution

If an adversary knows the time, which is possible in practice, then he/she can derive the key.
If the adversary knows only the approximate time, still he/she can exhaustively search all possible times.
Even if the adversary knows nothing about the time, it is still possible to brute force the variable s. This is since `int` data type in C is only either 2-byte (16-bit) or 4-byte (32-bit) long depending on the platforms used.

5. (*Wrong implementation leading to predictable pseudo random numbers.*) What is the different of using `Java.security.SecureRandom` or `/dev/urandom` compare to the the random number generator in question 4? Bob again realised his mistake. In his most updated version, the seed s is generated using the "secure" random number generator. The hash function `SHA1` is then similarly applied on s to get k and v. Does Bob use a correct mechanism to generate a good seed now?

   Unfortunately, there is still a vulnerability in Bob's implementation. Give a ciphertext-only attack that finds k?

   Solution

   Yes, Bob employs a correct method of generating the seed s, and subsequently the IV v.
   (*Note*: Please read about `Java.security.SecureRandom` and `/dev/urandom`.)

   Bob's implementation, however, insecure since an attacker can still find the key k used. Notice that the key k is derived by applying SHA-1 to the 160-bit $x_1 = v \,||\, r$, where v is the 128-bit IV and r is a 32-bit string. Since the attacker knows the IV v, then he/she will just need to guess the generated r. Given that r is only 32 bits, the attacker is therefore able to exhaustively search r. For each r, construct $x'_1 = v \,||\, r$, and compute $x'_2 = \texttt{SHA-1}(x'_1)$. Then, test whether the first 128-bit of x'_2 is the correct key k.

6. (*Private key is not the only information to be protected.*) Bob believes that he has discovered a simple yet secure public key scheme, which he named BC1 (Bob Cipher 1). It employs only a hash function like SHA-256, and a symmetric key encryption scheme like AES. (Note that `SHA-256` is a hash function in the family of `SHA2`, which produces 256-bit digest.) The scheme works as follows.

   The private key of of BC1 is a randomly chosen 320-bit string k, and its public key is a 256-bit $w = \texttt{SHA-256}(k)$.

   - Encryption: given the public key w and a plaintext x, employ AES to encrypt x with w as the 256-bit encryption key.
   - Decryption: given a ciphertext c and the private key k, compute $w = \texttt{SHA-256}(k)$, and then decrypt c using w.

   Bob made this statement: *"Note that* `SHA-256` *is believed to be one-way, and hence it is difficult to derive the private key k from the public key* w=`SHA-256`(k). *Therefore, the public-key scheme is secure."*

   Explain to Bob why BC1 is terribly insecure, and why his statement above is logically wrong.

(*Hint*: Refer to the security requirements of a public-key scheme, which are listed on slide 6 of Lecture 3.)

Solution

Bob is wrong since an adversary can simply use his public key w to decrypt the ciphertext c and obtain the plaintext x.

Bob is right in claiming:
S1: It is difficult to get the private key from the public key.
However, there is another security requirement:
S2: It is difficult to get the plaintext from the public key and the ciphertext.
Note that S2 $\Rightarrow$ S1, but S1 $\not\Rightarrow$ S2.
(*Also note*: Many people argue that RSA is secure just because of S1, which is an incomplete justification.)

7. (*Pitfalls: Authenticity vs Confidentiality*) Let us consider the penguin example in Lecture 3 slide 12. Bob is concern on both integrity and confidentiality. He chooses two secret keys k_1 and k_2. Given a plaintext, which is divided into n blocks $\langle b_1, b_2, \ldots, b_n \rangle$, the protection is done as following.

   (S1) Encrypt the plaintext using AES under CBC mode, with k_1 as the key. Let c be the AES ciphertext.[1]

   (S2) For each block b_i, compute $t_i = HMAC_{k_2}(b_i)$. That is, t_i is the mac for the block b_i, for each i.

   (S3) Computes $t_{n+1} = HMAC_{k_2}(\mathtt{n})$, where $\mathtt{n}$ is the string representation of the integer n.

   (S4) The final output is $\langle c, t_1, t_2, \ldots, t_n, t_{n+1} \rangle$.

   On the receiving end, an entity who knows k_1 and k_2 can carry out the following:

   (V1) Decrypts c to obtain $b_1, b_2, \ldots, b_n$.

   (V2) Verifies the value n using t_{n+1}. Rejects if not valid.

   (V3) Verifies block b_i using t_i for each i. Rejects if any one of them is not valid.

   Now, answer the following:

   (a) Consider an attacker who attempted to swap the 2nd and 3rd blocks in the plaintext. The attacker captured $\langle c, t_1, t_2, \ldots, t_n, t_{n+1} \rangle$ and replaced it by $\langle c', t_1, t_3, t_2, t_4, \ldots, b_n \rangle$, where c' is c but with the 2nd and 3rd block swapped. Would such tempering be detected?

[1] Here, c contains the IV.

Solution

Yes. Let $\langle \widetilde{b}_1, \widetilde{b}_3, \widetilde{b}_2, \widetilde{b}_4 \ldots \rangle$ be the decrypted plaintext on the swapped ciphertext. It is unlikely that $\widetilde{b}_3$ is equal to b_3. Thus the mac computed from $\widetilde{b}_3$ is unlikely equal to t_3. So, it likely to be rejected.

(b) Why it is necessary to have t_{n+1}?

Solution

Suppose the scheme doesn't include t_{n+1}. An attacker could claim that a truncated message that contains 1 block is the message, and it will be accepted.

(c) Why this scheme is flaw? (consider "confidentiality")

Solution

If $b_i = b_j$, then $t_i = t_j$, for any i, j. So, the leakage is similar to the leakage of ECB mode.

8. (*Concepts: Security Reduction.*) Lecture 3 states that a security requirement of cryptographic hash is "collision resistant". That is, it is difficult to find m_1, m_2 such that $h(m_1) = h(m_2)$ and $m_1 \neq m_2$.

   Here is an implied requirement, known as *one-way*. A function is *one-way* if, given x, it is computationally difficult to find a m such that $h(m) = x$.

   (a) Suppose we know a fast algorithm $\mathcal{A}$ that, when given x, successfully finds a m s.t $h(m) = x$. Show that we can construct another fast algorithm $\widetilde{\mathcal{A}}$ that can successfully find a collision with high probability (i.e. find m_1, m_2 such that $h(m_1) = h(m_2)$ and $m_1 \neq m_2$). You can treat $\mathcal{A}$ as a subroutine that can be called by $\widetilde{\mathcal{A}}$.

Solution

Consider this algorithm $\widetilde{\mathcal{A}}$:

i. Randomly pick a m. Compute $x = h(m)$.

ii. Run the fast algorithm $\mathcal{A}$ and taking x as input. Let m' be the output of $\mathcal{A}$.

iii. If $m' \neq m$ then outputs m', m and halts. Otherwise, goto step (i).

Note that

i. With high chance (probability ≥ 0.5), m' is not equal to m.

   To see that, let's suppose the pre-image of x contains m_1 and m_2, and the $\mathcal{A}$ picked m_1 as the output. Since step (i) randomly picks a message, the probability that the message chosen happen to be m_1 is not more than 0.5.

ii. If $\mathcal{A}$ is fast, so is $\widetilde{\mathcal{A}}$.

(b) Now, using the above, argue that the following statement is true: if a hash function is collision-resistant, then it is also one-way? (i.e. *collision resistant* $\Rightarrow$ *one-way*.)

Solution

Note that by definition, the fact that *there is an efficient way to find collision* is equivalent to the fact that *the scheme is not collision-resistant*.
Similarly, the fact that there is an efficient way to invert is equivalent to the fact that the scheme is not one-way.
In question (a), we shown that not *one-way* implies not *collision resistance*. Logically, this is equivalent to *collision resistance* implies *one-way*.
(Remark: the above is an intuition. The actual formulation requires more work.)

(Remarks: (1)We haven't formally define the meaning of "fast algorithm", "one-way", "computationally difficult", etc. Hence, the above is more of an intuition instead of formal proof. (2) This question illustrates the concept of security reduction.)

9. (*Padding oracle attack is practical*) Search the CVE database for a known vulnerability that is based on AES-CBC padding oracle attack. (note: there are also padding oracle attack on other encryption scheme).

Solution

Quite a few. Example, CVE-2020-8911. Search for "padding oracle" in `https://cve.mitre.org/cve/search_cve_list.html`

10. Find out more about these:
Single Sign-On (SSO), hardware random number generator, quantum random number generator, Authenticated-encryption, retinal vs iris scan.