CS2107 Tutorial 4 (Data-Origin Authentication)

School of Computing, NUS

13–17 September 2021

1. (*Birthday paradox:*) Suppose there are **at most** 100,000 hair glands on a human's scalp. Different persons have different numbers of hair glands (and we can assume that the numbers follow a uniform distribution).

   Now, suppose there are 1,000 undergraduate students in SoC. Are the chances high that there exist two SoC students with 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.
   (*Note*: In fact, the probability will be higher if we assume that all the students have at least N hair glands as you would normally expect.)
   (*Extra*: 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 a stream cipher always uses a randomly and uniformly chosen 64-bit IV when encrypting a plaintext into a ciphertext. In a set of collected 2^{33} ciphertexts, determine whether the probability that there exist two ciphertexts with the same IV is 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. (*Insecure usage of hash function:*) Cryptographic hash functions, such as `SHA-1`, are often employed to generate "pseudo-random" numbers. Given a short binary string s, which is also known as the *seed*, we can generate a pseudorandom sequence $x_1, x_2, x_3, \ldots$, where each x_i is a 160-bit (20-byte) string, as follows:

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

   Bob implemented a security protocol, which required a random 128-bit string k to serve as the AES encryption key, and a random 128-bit string v to serve as the IV. 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-1 hash function 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) pseudo random number generation:*) Consider the same scenario given in Question 3. Bob realized his mistake, and he changed his protocol. The updated program chose the seed s by using the following code snippet (similar to Slide 73 of Lecture 1 Part 3):

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

After the seed s was set, Bob followed the same steps described in Question 3 to generate x_1, x_2, and then derive the 128-bit v and 128-bit k.

If you are not familiar with C, the above C code can be replaced with a similar Java code that utilizes `java.util.Random` as mentioned in Lecture 1.

Explain why the above mechanism is still not secure by giving an attack that can 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.) In your solution, clearly state the information that the adversary has access to.

Solution

If an adversary knows the time, which is possible in practice, then he/she can derive the key.
Otherwise, 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. (*(More secure) pseudo random number generation, yet insecure protocol:*) What is the difference of using `Java.security.SecureRandom` or `/dev/urandom` compared to the the random number generator used in Question 4? Bob again realized his mistake. In his most updated version, the seed s is generated using a more secure random number generator shown on Slide 73 of Lecture 1 Part 3. The hash function `SHA-1` is then similarly applied on s to obtain k and v.

   Does Bob use a correct mechanism to generate a good seed now?

   Still, can you do a ciphertext-only attack that finds k? Suggest an algorithm to find the key k used.

   Solution

   Yes, Bob now employs a correct method of generating the seed s, and subsequently the IV v.
   (*Note*: Please read about `Java.security.SecureRandom` and `/dev/urandom`. For the latter, you can run `$ man 4 urandom` on your Ubuntu host.)

   The protocol is, 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 as discussed in Tutorial 1. 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. (*Insecure public-key scheme:*) 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 secret-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 secret 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=*$\texttt{SHA-256}(k)$*. 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 requirement of a public-key scheme mentioned on Slide 10 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. (*Enryption with mode-of-operation and integrity with MAC:*) Let us consider again the penguin example shown in "Issue 2: ECB with RSA" of Lecture 3. Let $b_1, b_2, b_3, \ldots$ be the blocks of the plaintext. Suppose that Bob is now concerned with both confidentiality and integrity. Hence, he chooses two secret keys k_1 and k_2. The penguin image is encrypted using AES under the CBC mode-of-operation using k_1 as the secret key. For integrity, for each block b_i, a MAC t_i is computed using HMAC with k_2 as the key. That is, the MAC $t_i = HMAC(b_i, k_2)$. Now, the final ciphertext consists of the outputs of AES and all the MAC values.

   Can the proposed scheme's "confidentiality" and "integrity" be compromised?

Solution

The scheme's **confidentiality** can be compromised since:

- There is no IV used in Bob's AES-CBC usage. Given the use of CBC mode-of-operation, two identical *plaintext blocks* will result into two different *ciphertext blocks*. However, if the AES-CBC is not randomized with an IV, a *certain plaintext* will always be encrypted into the same *ciphertext* even under different encryption instances. As discussed in the lecture, this is not desirable. Under a ciphertext-only attack, the attacker at least knows that the ciphertext comes from the same plaintext. Under a known-plaintext attack, the attacker can easily determine the plaintext from an observed known ciphertext, i.e. when the corresponding plaintext-ciphertext pair is already known by the attacker.
  (Furthermore, in general, a deterministic cipher is more susceptible to cryptanalysis under a chosen-plaintext attack scenario.)
- Despite the use of AES-CBC, the attacker now can additionally observe the *MACs of two plaintext blocks* to infer about these *plaintext blocks*. The attacker can know if two plaintext blocks are identical simply from matching the MACs of these two blocks.

Its **integrity**, meanwhile, can be compromised since the attacker can reorder the ciphertext blocks and respective MACs. Notice that, in CBC decryption, an altered/corrupted ciphertext block affects the corresponding plaintext block and the following plaintext block, but the rest of the blocks remain intact (see also `https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation`). As such, the effect of a block-ordering attack is *localized*.

In such a block-reordering attack, if the plaintext has some embedded extra information about the block order, or if out-of-order blocks can be observed "semantically" by the receiver, then the reordering attack can be easily detected. Yet, the attack can go undetected if the plaintext represents an image or video, where a small localized modification may *not be sufficiently noticable* by the receiver.

8. (*Security requirements of a cryptographic hash function:*) Lecture 3 states two security requirements of a cryptographic hash function $h()$, namely *collision resistant* and *preimage resistant* (*one way*). The former says that it is difficult to find m_1, m_2 such that $h(m_1) = h(m_2)$ and $m_1 \neq m_2$. The latter says that, given x, it is computationally difficult to find a m such that $h(m) = x$.

   Show that if a hash function h is collision resistant, then it is also one-way (i.e. *collision-resistant(h)* $\Rightarrow$ *one-way(h)*).

   (*Hint:* You can prove the implication statement above by showing that its contrapositive is true. That is, $\neg$ *one-way(h)* $\Rightarrow \neg$ *collision-resistant(h)*. This contrapositive

basically says that if it is easy to invert $h()$, then it is also easy to find a collision. This can be established as follows. Suppose there exists a fast algorithm $\mathcal{A}$ that, when given x, successfully finds a m such that $h(m) = x$. Given $\mathcal{A}$, then there also exists another fast algorithm $\widetilde{\mathcal{A}}$ that can successfully find a collision with high probability, i.e. $\widetilde{\mathcal{A}}$ can find m_1, m_2 such that $h(m_1) = h(m_2)$ and $m_1 \neq m_2$. Now, suggest how we can construct $\widetilde{\mathcal{A}}$ from $\mathcal{A}$!)

Solution

You can use the following probabilistic algorithm to construct $\widetilde{\mathcal{A}}$ (from $\mathcal{A}$):

- Randomly pick a m;
- Compute $x = h(m)$;
- Compute the inverse $m' = \mathcal{A}(x)$;
- If $m' \neq m$, then return(m, m'), else repeat;

There is a good probability that the algorithm above will soon stop, since:

- The set of hash values is finite;
- The set of messages is much larger than the set of hash values. Therefore, many different messages would be hashed into a single hash value, i.e. collisions should occur frequently.

(*Note:* If you want a more complete explanation, you can refer to Douglas R. Stinson, *Cryptography: Theory and Practice*, 3rd edition, Section 4.2.3.)

9. (*Miscellaneous:*) Find out more about these security terms:
Single Sign-On (SSO), hardware random number generator, quantum random number generator, retinal vs iris scan.

Solution

Please google/wiki the terms.

Be aware of the differences between pseudo random number generators and hardware random number generators by reading, for instance, `https://en.wikipedia.org/wiki/List_of_random_number_generators`.