CS2107 Tutorial 3 (Data-Origin Authentication)

School of Computing, NUS

9–13 September 2019

1. (*Birthday paradox:*) The are about 100,000 hair glands on a human's scalp, and different persons have different numbers of hair glands. Suppose there are 1,000 undergraduate students in SoC. Is the chance high that there exist two SoC students with the same number of hair glands?

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.

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.

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 47 of Lecture 1 Part 2):

```
#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.

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 47 of Lecture 1 Part 2. 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.

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=*`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.)

7. (*Enryption with mode-of-operation and integrity with MAC:*) Let us consider again the penguin example on slide 17 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 $t_i = \mathit{HMAC}(b_i, k_2)$. 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?

8. (*Security requirements of a cryptographic hash function:*) Slide 24 of Lecture 3 states that a security requirement of cryptographic hash function 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 a requirement implied by collision resistant, which is known as *one-way* (*preimage resistant*). A function $h()$ is *one-way* if, 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 $\mathcal{A}$!)

   (Remarks: We haven't formally defined the meaning of "fast algorithm", "one-way", "computationally difficult", etc. Hence, the above inference is more of an intuition instead of a formal proof. However, it will help you understand more about the security requirements of a cryptographic hash function.)

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