C2107 Tutorial 4 (Hash, mac)

Chang E.-C., School of Computing, NUS

September 8, 2025

Note:

- We assume it is feasible to carry out 2^{64} operations.

1. (*Pitfalls: Cryptographic hash is "cryptographically pseudorandom" but not "truly random"*) Cryptographic hash functions 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 pseudorandom sequence $x_1, x_2, x_3, \ldots$ in the following way.

   - Let $x_1 = \mathtt{H}(s)$, and $x_{i+1} = \mathtt{H}(x_i)$ 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 usages. It is important that the k remains secret.

   Let's assume that the above protocol is part of a standard, and the standard explicitly states that the k has to be randomly generated.

   Bob implemented the above component. To choose the k and v in step (a), Bob sets the seed s to be a string of 160 zeros, and then obtained x_1 and x_2 as described above. Here we assume that the hash H() is collision resistant and it outputs 160-bit digest. 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 H *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.

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

   Consider the same scenario in question 1. Bob realised his mistake and changed his program. The updated program chose the s by using the following.

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

After the seed s was chosen, following the same step in question 1, Bob applied `H`, 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`.

Explain why the above is not secure by giving a ciphertext-only attack that obtains 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.)

(Hints: There are two different ways of attack. The first way needs to know the time. The second way does not need to know the time. For the second way, note that the `int` type, depending on OS, is either 16 or 32 bits.)

3. (*Wrong implementation leads to leakage of the secret key.*) What is the different of using `Java.security.SecureRandom` or `/dev/urandom` compare to the the random number generator in question 2? Bob again realised his mistake. In his most updated version, the seed s is generated using the "secure" random number generator. The hash function `H` is then similarly applied on s to get k and v.

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

   (Hint: The IV is derived from the key, and the IV is made public.)

4. (*Online vs offline attack*)[1]

   You are assessing a password login system. The user keys in the userid and password into the client device. Next, the server and client carried out a password verification protocol. At the end of the protocol, the server will either accept or reject, and the client will know the outcome.

   The server is able to carry out verification at a very high rate (say, 10,000 per second), but it mitigates exhaustive search in two ways: (1) Each verification process is intentionally delayed by 1 second; (2) Any user (identified by userid) can carry out at most one verification at any one time (hence, an attacker cannot login from 2 different ip-address to try two different passwords.)

   (a) The system recommends having password of at least 10 alphanumeric (including uppercase and lowercases) characters. Based on the guideline by RFC4086 in Tutorial 3, is 10 sufficient?

[1]The protocol contains some essence of WPA-PSK 4-way handshake protocol. WPA-PSK is commonly deployed in home wifi access point. Offline dictionary attack can be carried out on WPA-PSK. `https://cylab.be/blog/32/how-does-wpawpa2-wifi-security-work-and-how-to-crack-it`. Tools like `aircrack` employ offline guessing.

(b) The verification protocol is quite complex and can be summarised as below: Let's call the client the *prover* and the server the *verifier*. In this question, for simplicity in analysis, we assume that verifier is authentic and do not consider scenario where an attacker pretends to be the verifier.

   i. $(P \to V : u)$. The prover gets userid u and password p from the (human) user. Next, the prover sends u to the verifier.

   ii. $(P \leftarrow V : c)$. From u, verifier lookups the password p from the password file. Verifier derives a 128-bit key from p by

   $$k = \texttt{SHA3}(p).$$

   The verifier randomly generates a 128-bit string r and sends the ciphertext

   $$c = \texttt{ENC}(k, r)$$

   to the prover, where $\texttt{ENC}()$ is some secure encryption scheme, e.g. AES in CBC mode[2].

   iii. $(P \to V : h)$. The prover uses p to get k. Next from c, the prover can get r. Prover computes

   $$h = \texttt{mac}(k, r)$$

   and sends h to the verifier, where $\texttt{mac}$ is some secure mac scheme, e.g. HMAC.

   iv. (V accept/reject). The verifier, using its own k, r, check that the received h indeed satifies

   $$h = \texttt{mac}(k, r).$$

   If so, accepts that the prover is authentic and declare "accept"; otherwise, "reject".

The protocol is to be carried out wirelessly. So, in a reasonable attack model, we should assume presence of an eavesdropper, i.e. the attacker can get c, u, h.

- Derive an "offline attack" and discuss whether 10-character p is sufficient.
- Suppose there is an implementation flaw and r is also fixed with value 0....0. What is the implication?
- For typical home wifi access point, how many characters (assuming they are randomly chosen) are required?

5. (*Design of mac*) It is tempting to design a mac using encryption. Given a message m and key k, the mac is $\text{Enc}_k(H(m))$ where $H(\cdot)$ is a collision

[2]Any scheme that is secure from chosen plaintext attack would do. AES in CTR mode would also do.

resistant hash, and $\text{Enc}_k(\cdot)$ is a symmetric key stream cipher, for instance AES CTR mode.

Explain why it is not a secure mac.