C2107 Tutorial 5 (Authentication and Certificate)

School of Computing, NUS

September 16, 2025

1. (*Hash Design*) *This question illustrates the difficulty in designing hash and further illustrates why it is not advisable to make changes to standardised crypto.*

   Bob designed a new hash function H based on existing `SHA3` in this way:

   $H(x) = \texttt{SHA3}(x) \oplus \texttt{SHA3}(\texttt{rev}(x))$ where `rev` is the string reverse function on the binary sequence x, where the last bit become the first bit. E.g. `rev`(101000) become 000101.

   Give a collision to show that H is not collision resistant. In your answer, the length of the input should be at least 8.

2. (*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. What is the probability that there exists two SoC students having the same number of hair glands?

3. (*Birthday attack*) Suppose the length of IV is 64 bits. During each encryption, the IV is randomly and uniformly chosen. In a set of 2^{32} ciphertext, what is the probability that there are two ciphertext with the same IV? What is the respective probability when the number of ciphertexts is $2^{31}, 2^{33}, 2^{34}, 2^{35}$?

4. (*Birthday attack: two-pool variant*)

   (a) To do next question, a variant of Birthday attack would be useful. The version in lecture note takes a pair from a common pool. Here, there are two pools. One item is drawn from the first pool, and another item is drawn from the another pool. Below is an useful theorem.

   Let $\mathcal{S}$ be a set of k distinct elements where each element is an n bits binary string. Let us independently and randomly select m n-bit binary strings and put them in the set $\mathcal{T}$. The probability that $\mathcal{S}$ has non-empty intersection with $\mathcal{T}$ is more than

   $$1 - 2.7^{-km2^{-n}}$$

   (b) Consider this scenario. A machine V sends out a query to a server. The query contains a 16-bit id. The server replies with an answer and the answer contains the 16-bit id. V may send out multiple queries, and would receive multiple answers from the server. To resolve the

ambiguity, V would match the 16-bit id in the query and answer. An answer that doesn't have matching id will be dropped.

Now, we have an attacker. The attacker cannot see the content of the query, but know the fact that a query being sent. The attacker can forge an answer and send to V. However, note that V accepts an answer only if its id matches with one of the queries. So the chance that the attacker succeed is low. One way for the attacker to increase the chance is by forging large number of answers with randomly chosen 16-bit ids, hoping that one of them matches a query.

Suppose the attacker detected that V has sent out 128 queries. How many answers should the attacker forge so that with high probability (greater than 0.5) the attack succeed? Give the minimum number of answers required. (Hint: the answer is approx 512 which is $2^{16}/2^7$.)

5. **Single Signed On (SSO).** When a user is using a new laptop A to visit a website, say Facebook, the website typically would prompt the user to manually key in the userid and password. However, for subsequent visits using the same laptop, the website does not prompt the user for the password. E.g. when the user wants to post a message to Facebook, there is no further prompt for userid & password[1]. This is very convenient as the user only needs to *manually* login once, and hence the term "Single Signed On (SSO)". With SSO, the user can easily upload photo to Facebook or other social media platform.

   Although the user doesn't manually login, the laptop automatically carries out authentication with the server. The following is a way to achieve SSO:

   S1. After a user with userid u successful manual login, the web server generates a string t called *authentication token*[2], and sends t to the laptop A.

   S2. For each subsequent visit, the laptop A automatically sends (t, u) to the server without the user involvement. If the authentication token is correct, the website accepts and does not prompt the user.

[1] We assume that the client had already conducted unilateral authentication with the server and a secure channel is already established. Here, the client is sure that the server is authentic, whereas the server does not know the authenticity of the client and thus ask for the password. This is a typical setting in web application. E.g. The client first establishes HTTPS with a bank server through unilateral authentication. Next, the bank server asks the client to manually key in password.

[2] In web, the token is sent as "cookie", which will be covered later in the topic of web security.

Threat Model. We assume that an attacker has the following capabilities:

- The attacker cannot sniff the communication between its victims and server.
- The attacker has the valid tokens of a few userid (but not the victim's token). This assumption is reasonable because the attacker can register for a few accounts and obtains the corresponding valid tokens.
- As usual, we assume the attacker is aware of the algorithm.

Attacker's goal (forgery) : Given a victim userid u_0, find a valid token t_0.

Now, answer the following questions:

(a) An authentication token typically has an expiry date.Why?
(Hints: Imagine an unfortunate scenario where the token is stolen without knowledge of the owner. In such scenario, what is the difference of having or without a expiry date?)

(b) Here is an insecure design of the tokens. The server uses $t = \langle m, y \rangle$ as the token, where

- $m = d \| r \| u$;
- $y = H(m)$;
- d : date and time (precision upto seconds) of the token creation represented as ASCII string;
- r : a randomly selected 128-bit string, which is to be served as salt;
- u : 10-character user id;
- $H(\cdot)$: a collision resistant hash.

After the server received t, it accepts iff (1) the userid is in the correct format; (2) the token not yet expired; and (3) $H(m)$ is indeed equal to y.

i. Let us consider an attacker, who knows the userid of Alice and wants to login as Alice. Show that the above is not secure. Is there a"secret" in the above?

ii. The vulnerability can be fixed by using mac instead of hash. Give such construction. Let's call this the *mac-based variant.*

iii. (Optional. Reduction proof) Show that the mac-based variant is secure. (Hints: This requires a few steps.

A. First, show that if an attacker is able to find a valid token, then the attacker has a way to "break" the mac scheme by forging a mac.

B. Next, argue by contradiction that the mac-based method is secure.)

(c) Here is another server. This server keeps a database of userid, each associated with a token. After a user u successfully manually logged in, the server generates a random 128-bit r as token. The tuple (u, r)

is updated into the database $\mathcal{D}$ and the r is sent to the user. To verify the server simply checks $\mathcal{D}$. Let's call this method the *random token variant*.

i. Argue why the random token variant is secure.

ii. The mac-based variant is more commonly used. Give a reason (based on performance and usability) why the mac-based is preferred over the random token version.

6. (*Illustration on security reduction proof*) Show that, given a randomly chosen binary string h, it is computationally difficult to find a message x s.t. $H(x)$ is a substring in h, where $H(\cdot)$ is a collision resistant hash.

   Note: The proof is optional, but this fact is required and useful in other part of this course.

   (Hint: The main idea follows this argument. Suppose there is fast method A to do above, we can design another algorithm $\widetilde{A}$ that uses A as sub-routine to solve the one-way problem. This leads to contradiction because one-way is hard. Hence, there isn't such a fast A.)