CSC 591 Cryptography 10/31/2018

Lecture 18: Practice with Hash Functions and MACs

Lecturer: Alessandra Scafuro *Scribe: Srija Ganguly*

1. Hash Function from Discrete Logarithm Assumption

We have been introduced to Hash Functions and the Discrete Logarithm Assumption before. This question leads to the application of the latter to build a Hash Function.

1.1 Defintion

Let us revisit the definition of Discrete Log Assumption in simple words.

For a cyclic group G of the order q and a generator g, let there be an element x that belongs to G such that $y = g^x$. The Discrete Log Assumption states that it is hard to find such a value x which is also known as the Discrete Log of y. In other words, for a Probabilistic Polynomial Time (PPT) algorithm A , the probability of finding a Discrete Log of y in G is a negligible function in n:

$Pr[DiscreteLog_{A,G}(y) = 1] <= Negl(n)$.

1.2 Assumption

The assumption in the following questions the Discrete Log Assumption. Let G be a cyclic group in which the Discrete Log Assumption holds, with generator g and order q.

1.3 Scheme

Let there be a Hash Function H_{DL} that takes in two inputs (x_1, x_2) such that $x_1, x_2 \in G$ and outputs an element that also belongs to the group.

The scheme will be as follows:

Gen (1^n) (assume Gen has input G, g, q)

- Pick a random element $h \leftarrow G$.
- Output h.

Hash: $H^s_{DL}(x_1, x_2)$ (where x_1 and x_2 are two inputs of G)

- Parse $s = g, h$.
- Output $z = g^{x_1} \cdot h^{x_2}$

1.4 Security Proof

Theorem: If the Discrete Logarithm Assumption holds in G then the hash function H_{DL} is Collision Resistant.

Proof: Towards a contradiction let us assume there is a PPT adversary A_{coll} that is able to output a collision for function H_{DL} on input g, q, h with probability $p(n)$.
Thus the reduction idea is that A_{coll} can be used to find the discrete log of h in base g, thus contradicting the assumption that the discrete logarithm problem is hard to solve in G.
Let A_{coll} input (x_1, x_2) and (x_3, x_4) which form a collision on Hash output. This would mean $H^s_{DL}(x_1, x_2) = H^s_{DL}(x_3, x_4)$. Then it also holds that :
$g^{x_1} \cdot h^{x_2} = g^{x_3} \cdot h^{x_4}$
Now , h can be expressed as g^{α} for some $\alpha \in G$.
$=> g^{x_1} \cdot g^{\alpha * x_2} = g^{x_3} \cdot g^{\alpha * x_4}$
$=> g^{x_1 + \alpha * x_2} = g^{x_3 + \alpha * x_4}$
$=> DiscreteLog(g^{x_1 + \alpha * x_2}) = DiscreteLog(g^{x_3 + \alpha * x_4})$
$=> x_1 + \alpha \cdot x_2 = x_3 + \alpha \cdot x_4$
$=> \alpha = \frac{x_3 - x_1}{x_2 - x_4}$

Thus the discrete logarithm of h is α. Just by knowing the elements, we are able to find the discrete logarithm which is contradicting the assumption that it is hard to compute in G. Therefore the probability for A_{coll} to input the two sets of inputs that finds a collision in the Hash Function is negligible.

The Hash Function H^s_{DL} is Collision Resistant.

2. Practice with MAC and Hash

We have been introduced to the concept of MAC earlier. This exercise has two questions that aim to apply the concept of MAC to generate Hash Functions.

2.1. Insecure MAC

Following is a MAC construction which is declared insecure. The aim is to write an adversary that breaks the unforgeability game.

2.1.1 Assumption:

The function F is a Pseudo Random Function (PRF).

2.1.2 Scheme:

$\underline{\pi Gen(1^n)}$

- Output a random key $k \in \{0,1\}^n$.

$\underline{\pi.\text{MAC}(m_1, ..., m_l)}$: where $m_i \in \{0,1\}^n$

- Output tag $t := F_k(m_1) \oplus .....F_k(m_l)$

2.1.3 Attacks:

Since we know that the given MAC is not secure, we come up with an attack where a PPT adversary A_{MAC} tries to break the unforgeability game $MAC - Forge(A_{MAC}, \pi, n)$. We need to remember the basic that in this case, the A_{MAC} doesn't have access to the PRF but only to the game that accesses the Oracle instantiated with either PRF or a Truly Random Function. There may be multiple valid attacks. Here two such examples are shown.

In one version of the attack the A_{MAC} works as follows:

1. Query
   - $m_1 \to t_1$
   - $0^n \to t_2$
2. Output
   - message $m = (0^n, m_1)$
   - tag $t = t_1 \oplus t_2$

Analysis: The output provided by A_{MAC} is valid because $m \neq m_1$ and $m \neq 0^n$. The tag $t_1 = F_k(m_1)$, tag $t_2 = F_k(0^n)$ and the tag $t = t_1 \oplus t_2$ which can be written as $t = F_k(m_1) \oplus F_k(0^n) = F_k(m)$.

Therefore, for the messages m_1 and m_2, A_{MAC} can easily know the tag for m if it matches the xor of the tags of his query messages. This would lead him to break the game.

In another version of an attack, let the A_{MAC} work as follows:

1. Query
   - $m_1 = (x_1||x_2) \to t_1$
     Here message is a congregation of two messages x_1 and x_2.
2. Output
   - message $m = (x_2||x_1)$
   - tag t

Analysis: The output provided by A_{MAC} is valid because $m \neq m_1$. The tag $t_1 = F_k(m_1) = F_k(x_1) \oplus F_k(x_2)$.
The tag $t = F_k(m) = F_k(x_2) \oplus F_k(x_1)$.

Therefore the output tag matches queried tag and the A_{MAC} breaks the game. Here the attack is even more efficient because there is just one query required.

2.2. Another MAC construction

Following is a MAC construction whose security is not known. We have to prove if the MAC is secure. If its insecure ,only an attack is needed.

2.2.1 Assumption:

The function F is a PRF in the given MAC construction.

2.2.2 Scheme:

$\pi Gen(1^n)$

- Output a random key $k \in \{0,1\}^n$.

π.MAC$(m_1, ..., m_l)$: where $m_i \in \{0,1\}^n$

- Output tag $t := F_k(1||m_1) \oplus .....F_k(l||m_l)$

2.2.3 Attacks:

At first look it may look secure. For proving that a scheme is insecure, we need to find just one attack that breaks the unforgeability game. It is advised to try several attacks with small messages till you are satisfied that there are no possible attacks here. The scheme takes several messages of n bits together to output a tag that makes use of the position of a message m_i. The intuition behind the attack is to find a common message amongst several query messages that have the same position in a new message that is output by the adversary.

After several attempts, we come up with an attack where a PPT adversary A_{MAC} tries to break the unforgeability game $MAC - Forge(A_{MAC}, \pi, n)$. We need to remember the basic that in this case, the A_{MAC} doesn't have access to the PRF but only to the game that accesses the Oracle instantiated with either PRF or a Truly Random Function.

In the attack A_{MAC} works as follows:

1. Query
   - $m_0 \to t_0$
   - $m_1 \to t_1$
   - $(m_1, m_2) \to t_2$
2. Output
   - message (m_0, m_2)
   - tag $t = t_0 \oplus t_1 \oplus t_2$

Analysis: The output provided by A_{MAC} is valid because $(m_0, m_2) \neq m_0$ and $(m_0, m_2) \neq m_1$ and $(m_0, m_2) \neq (m_1, m_2)$. The tag $t_0 = F_k(1||m_0)$, tag $t_1 = F_k(1||m_1)$, tag $t_2 = F_k(1||m_1) \oplus F_k(2||m_2)$
The output tag $t = t_0 \oplus t_1 \oplus t_2 = F_k(1||m_0) \oplus F_k(1||m_1) \oplus F_k(1||m_1) \oplus F_k(2||m_2)$ which can be written as $t = F_k(1||m_0) \oplus F_k(2||m_2)$. This last expression becomes the same result when the output message (m_0, m_2) is passed through the MAC function.

Therefore, A_{MAC} is able to generate a valid tag for the messages he outputs, based on the messages he queries. This is able to make him break the game. Thus the MAC construction is not secure.