Assignment 3

Cristian Bassotto, Roberta Chissich

11th September 2024

(1a) Argue that once you find two different messages M, M' of equal size for which the Merkle-Damgard evaluations collide, you can freely construct many other collisions.
When we find two different messages M and M' of equal size that produce the same SHA-1 hash value, we can exploit the properties of the Merkle-Damgård construction to create many additional collisions, by employing the following approach:

1. SHA-1 processes input messages in fixed-size blocks (512 bits) and maintains a state that updates with each block. The final hash is derived from the last state after processing all blocks. (Merkle-Damgård Structure)

2. If M and M' yield the same hash, this means that, at some point in the processing, the internal state of SHA-1 (the value of the hash being calculated) became the same for both inputs.

3. The Merkle-Damgård design allows us to append additional data to both messages M and M' without altering the collision. This is due to the fact that the next state of the hash depends only on the previous state and the next input block, not on the specific input message.

For example, if we append a block B to both M and M', both $M\|B$ and $M'\|B$ will hash to the same value because they both start with the same state after M and M' are processed. This property can be used to generate an infinite number of collisions simply by varying the additional data we append.

(1b) How much work would it be to exhaustively find two unpadded messages of size 5 blocks (or 2560 bits) for which their Merkle-Damgard evaluations collide?
To find two unpadded messages of size 5 blocks (or 2560 bits) that collide, we need to consider the properties of SHA-1:

1. SHA-1 produces a hash output of 160 bits.

2. The message length is 5 blocks, where each block is 512 bits. Thus, the total size is:

$$5 \text{ blocks} \times 512 \text{ bits/block} = 2560 \text{ bits}$$

3. The birthday paradox suggests that to find a collision among messages that output a hash of n bits, the complexity is approximately $2^{n/2}$. For SHA-1, $n = 160$, so:

$$\text{Work required} \approx 2^{160/2} = 2^{80}$$

4. An exhaustive search involves trying every possible combination of 2560 bits, so 2^{2560}. However, since the actual search for collisions is constrained by the number of unique hash outputs in SHA-1, the complexity of finding two messages that collide through exhaustive search would be at most $2^{160} + 1$.

(1c) Compute SHA-1($S \parallel M$) and SHA-1($S \parallel M'$).

SHA-1($S \parallel M$) = f92d74e3874587aaf443d1db961d4e26dde13e9c

SHA-1($S \parallel M'$) = f92d74e3874587aaf443d1db961d4e26dde13e9c

(1d) Compute SHA-1($S \parallel M \parallel M''$) and SHA-1($S \parallel M' \parallel M''$).

SHA-1($S \parallel M \parallel M''$) = 4a010556637b3fe7001cb16198040d31eb7aa1d1

SHA-1($S \parallel M' \parallel M''$) = 4a010556637b3fe7001cb16198040d31eb7aa1d1

(1e) Find a prefix P for M_a and a second message M_b such that SHA-1($P \parallel M_a$) = SHA-1(M_b), where $P \parallel M_a$ and M_b are distinct and of equal size.

We can define $P = S \parallel M$ and $M_b = S \parallel M' \parallel M_a$. The two messages $P \parallel M_a$ and M_b are distinct because M and M' are different, ensuring that P and M_b are not equal. Furthermore, both messages are of the same size, since $S \parallel M$ and $S \parallel M'$ have the same length.

(1f) Compute the corresponding SHA-1 hash digest.

SHA-1($P \parallel M_a$) = SHA-1($S \parallel M \parallel M_a$) = 601117141fc3cbb6dcdea2d44db1095f1fc0e119

SHA-1(M_b) = SHA-1($S \parallel M' \parallel M_a$) = 601117141fc3cbb6dcdea2d44db1095f1fc0e119

(2a) What is the proven minimal complexity to find a collision against a sponge construction?

The minimal complexity to find a collision in a sponge construction is typically determined by the birthday bound. For a sponge construction that used an output size of n bits, the proven complexiy to find a collision is approximately $O(2^{n/2})$.

(2b) Sketch the generic approach to find a collision for sponge-based hash functions. (Note: you have to describe two attacks.)

1. Attack 1: The classic approach based on the birthday paradox, so it leverages the fact that the probability of two random inputs producing the same output increases significantly as the number of inputs grows.

   So, we (a) randomly generate a large number of input messages (e.g., using a range of small variations of a base input). (b) For each input, compute the hash value using the sponge construction. (c) Store each output in a hash table. If two inputs produce the same output, a collision has been found.

   The expected number of inputs that need to be hashed before finding a collision is approx. $O(2^{n/2})$ due to the birthday bound.

2. Attack 2: Another approach is the generic collision search.

   This attack involves systematically exploring the input space and observing the outputs. By carefully choosing inputs that manipulate the internal state of the sponge, an attacker can increase the chances of producing the same output for different inputs.

   The complexity of this attack is generally $O(2^{(c/2)})$, where c is the capacity of the sponge. This method can be particularly effective if the capacity is significantly smaller than the output length.

(2c) Explain how the attacks of question 2b can be transformed to a second preimage attack against the sponge construction.

A second preimage attack is an attempt to find a different input x' that produces the same hash output as a given input x. Unlike collision attacks, which focus on finding two distinct inputs that hash to the same output, second preimage attacks target a specific input's output.

So, the two attacks can be transformed into a second preimage attack by computing the hash $h(x)$ for the original input and storing it. Then by modifying x systematically to generate new inputs x', computing $h(x')$, and checking if $h(x') = h(x)$. The expected complexity remains around $O(2^n)$, since the attacker must find a specific input.

(2d) Give lower bounds for the values c and n if we require at least 128 bits of security against collision attacks.

This requires that the expected number of operations needed to find a collision is at least 2^{128}. Since that the complexity for finding a collision is approximately $O(2^{n/2})$, we need the requirement:

$$O(2^{n/2}) \geq 2^{128}$$

To satisfy the above inequality, we need:

$$\frac{n}{2} \geq 128 \quad \Rightarrow \quad n \geq 256$$

Thus, the output size n must be at least 256 bits.
The capacity c relates to the state size b of the sponge function:

$$c = b - r$$

where r is the rate (the portion of the state output). To maintain security, the capacity should also be sufficient to ensure that the sponge can absorb a large number of inputs without compromising security. For a good balance of security, it's often recommended that the capacity c should be at least half of the state size b and also at least equal to the output size n. Therefore, if we use $n = 256$, we can ensure that:

$$c \geq 256 \quad \text{(to achieve strong security)}$$

However, this also means that the state size b should be at least:

$$b \geq c + r \quad \text{(where } r \text{ can be chosen appropriately)}$$

(2e) Give an example parameter set (b, c, r, n) for which the sponge construction achieves higher generic second preimage resistance than collision resistance.
For example,

- b (bit length of the state): 512 bits
- c (capacity): 448 bits
- r (rate): 64 bits
- n (output length): 256 bits

The total size of the state is 512 bits. This state size encompasses both the capacity and the rate of the sponge construction.
The capacity is set to 448 bits. A higher capacity allows the sponge to absorb a significant amount of data without compromising security. Specifically, a larger capacity enhances resistance against collision attacks, as it effectively increases the amount of possible states that can be generated.
The rate is set to 64 bits. This means that for each input block processed, 64 bits of output can be produced. The rate influences how quickly data can be processed, but a smaller rate can enhance security.

The output size is set to 256 bits. This is a typical size for a cryptographic hash function output, providing a sufficient level of security for many applications.

Collision Resistance: The expected complexity to find a collision in a sponge construction is approximately

$$O(2^{c/2}).$$

For this parameter set:

$$O(2^{448/2}) = O(2^{224}).$$

This indicates that collision resistance is effectively 224 bits.

Second Preimage Resustance: The expected complexity to find a second preimage is given by

$$O(2^n).$$

In this case:

$$O(2^{256}).$$

This indicates that second preimage resistance is effectively 256 bits.

Thus, in this parameter set ($b = 512, c = 448, r = 64, n = 256$), the second preimage resistance $O(2^{256})$ is higher than the collision resistance $O(2^{224})$.

(3a) Describe an attacker that breaks the confidentiality of SpongeWrap in a constant number of queries.

The attacker has access to the SpongeWrap encryption scheme and can choose messages and nonces for encryption, and can repeat nonces for encryption queries.

The attacker selects a nonce N and encrypts two different plaintext messages M_1 and M_2 using the same nonce:

- The attacker queries the encryption oracle with (N, M_1) to obtain the ciphertext C_1.
- The attacker then queries the encryption oracle again with the same nonce N but a different message M_2 to obtain the ciphertext C_2.

Formally, the attacker has:

$$C_1 = M_1 \oplus K(N)$$

$$C_2 = M_2 \oplus K(N)$$

Where $K(N)$ is the key stream generated from the nonce N.

The encryption process in SpongeWrap typically involves mixing the nonce with the plaintext to produce the ciphertext. Given that the same nonce is used for both encryptions, the attacker can

observe the following:

$$C_1 = \text{Encrypt}(N, M_1)$$

$$C_2 = \text{Encrypt}(N, M_2)$$

Since the same nonce N is used, the attacker can derive a relationship between the two ciphertexts:

$$C_1 \oplus C_2 = (M_1 \oplus K(N)) \oplus (M_2 \oplus K(N)) = M_1 \oplus M_2$$

This operation does not depend on the key stream $K(N)$ because it cancels out.
By observing C_1 and C_2, the attacker can compute $M_1 \oplus M_2$. If the attacker knows or can guess one of the plaintexts (say M_1), they can easily recover the other plaintext M_2:

$$M_2 = C_1 \oplus C_2 \oplus M_1$$

If they don't know it beforehand, they still have $M_1 \oplus M_2$, which may reveal information about both plaintexts depending on their structure.
This effectively breaks the confidentiality of the SpongeWrap scheme, as the attacker can recover plaintexts from ciphertexts when nonces are reused.

(3b) Describe an attacker that breaks the authenticity of SpongeWrap in a constant number of queries. (Note: the distinguisher is not allowed to repeat nonces for encryption queries.)
The attacker has access to the SpongeWrap encryption scheme and can issue encryption queries with unique nonces. The attacker also has access to the decryption oracle, which allows them to query the system with known ciphertexts and receive the corresponding plaintexts.
The attacker can perform the following steps:

- The attacker selects a nonce N_1 and a plaintext message M_1 and queries the encryption oracle to obtain the ciphertext C_1:

$$C_1 = \text{Encrypt}(N_1, M_1)$$

- The attacker then selects a different nonce N_2 and a distinct plaintext message M_2 and queries the encryption oracle again to obtain the ciphertext C_2:

$$C_2 = \text{Encrypt}(N_2, M_2)$$

- The attacker can now create a new ciphertext C' using the ciphertext C_1 and the known plaintext M_2:

$$C' = C_1 \oplus M_2$$

We are manipulating the original ciphertext C_1 by XORing it with the known plaintext M_2.

- The attacker then submits the manipulated ciphertext C' to the decryption oracle:

$$\text{Decrypt}(C')$$

Since C' is constructed using valid ciphertext and known plaintext, the decryption oracle will return a plaintext M' that is inconsistent with the original plaintexts M_1 and M_2.

The attacker has successfully produced a valid decryption output M' from a manipulated ciphertext C', which was not generated through a legitimate encryption process.