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-Damgård 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 of capacity c and output size n, the proven complexity to find a collision is approximately $O\left(min(2^{c/2}, 2^{n/2})\right)$ due to the constrains of the sponge, given by its capacity, and of a RO, given by its output size.

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

1. Attack 1: Birthday Attack (External Collision Attack)

   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) where n is the output size.

2. Attack 2: Internal State Collision Attack

   This attack targets the internal state of the sponge construction, denoted as $S = R \parallel C$, where C is the capacity of size c, and R is the rate. During each iteration, a message block of length $|R|$ is XORed into R, followed by the application of the permutation P to the full state. The goal is to find a collision in C, which can be achieved in $2^{c/2}$ steps using a birthday attack. Once a zero difference in C is found, any difference ΔR in R can be canceled by appending an appropriate pair of message blocks. Thus, if (M_1, M_2) yields zero difference in C and difference ΔR in R, the pair $(M_1 \parallel X, M_2 \parallel (X \oplus \Delta R))$ will form a collision for the full function F for any message block X.

   The complexity of this attack is $O(2^{c/2})$, where c is the capacity of the sponge. This method can be effective if the capacity is 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 $h(x)$ or the internal state for the original input and storing them. Then by modifying x systematically to generate new inputs x', computing $h(x')$ or its internal states, and checking if $h(x') = h(x)$ or if internal states match.

The expected complexity is around $O(2^n)$ for the external attack, because it does not benefit from the birthday paradox, while the internal attack has a complexity $O(2^{c/2})$.

(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 $O\left(min(2^{n/2}, 2^{c/2})\right)$, we need to satisfy the requirements:

$$\begin{cases} 2^{n/2} \geq 2^{128} \\ 2^{c/2} \geq 2^{128} \end{cases} \implies \begin{cases} n \geq 256 \\ c \geq 256 \end{cases}$$

Thus, the output size n must be at least 256 bits as well as the capacity c.

(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: As seen before, the expected complexity to find a collision in a sponge construction is approximately

$$O(min(2^{n/2}, 2^{c/2})).$$

For this parameter set:

$$min(2^{n/2}, 2^{c/2}) = min(2^{256/2}, 2^{448/2}) = min(2^{128}, 2^{224}) = 2^{128}$$

This indicates that we have 128 bits of security against collision attacks

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

$$O(min(2^n, 2^{c/2}))$$

In this case:

$$min(2^n, 2^{c/2}) = min(2^{256}, 2^{448/2}) = min(2^{256}, 2^{224}) = 2^{224}$$

This indicates that we have 224 bits of security against second preimage attacks.
Thus, in this parameter set ($b = 512, c = 448, r = 64, n = 256$), the second preimage resistance $O(2^{224})$ is higher than the collision resistance $O(2^{128})$.

(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.
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 omission of domain separation bits (1/0) that distinguish between associated data (A) and message (M) opens the possibility for an attacker to exploit this vulnerability using a length extension attack. In particular, there are two possible attacks that will break the authenticity of the scheme. This two attacks are complementary, depending if the attacker wants to manipulate first the message or the associated data. In the following we will explain the attack manipulating first the message encrypted.

1. First we construct a message $M_1 = M_1' \parallel 10^* \parallel M_1''$ where 10^* is the padding that would be added normally inside the sponge if only M_1' was encrypted. This padding is crucial as it allows the attacker to control the length of the message being processed.

2. Now we can encrypt everything and receive:

$$(C_1, T_1) = \text{SpongeWrap}(N, A_1, M_1)$$

   where N is the nonce, A_1 the associated data and M_1 the previously constructed message.

3. Then we construct the new associated data and cyphertext for the forgery as follows:

$$A_F = A_1 \parallel 10^* \parallel M_1'$$

$$C_F = \text{last } |M_1''| \text{ bits of } C_1$$

   The basic idea behind is to consider the first part of the message as part of the associated data. By doing so, the combination of the associated data and the message absorbed by the sponge remains the same as before, resulting in the same tag T_1. However, the actual message that has been encrypted is now shorter.

4. Now we can procede with the forgery using the new associated data and cyphertext, along with the previously calculated tag:

$$M_F = \text{SpongeWrap}(N, A_F, C_F, T_1)$$

   where $M_F = M_1''$

(4) BONUS

We ensured that the SHA-1 evaluation of this PDF ends with "BADA55". We have implemented an algorithm in Python that uses a brute force approach to find the correct padding to add at the end of the file.

The code is available here.