

## INFORMATION SECURITY LAB.3

### A.Y 2024/2025

---

## Lab 3 - Information Security

### Cyberducks

Andrea Andreozzi (2163406)  
Riccardo Scalco (2155352)  
Sergio Cibecchini (2155353)  
Luca Ferrari (2166294)

---

## 1 Task 1 - Implementing the authentication scheme

In this task we had to implement a system for message authentication and integrity protection based on a symmetric key system.

Our system should safely transmit the message  $u$  of length  $M$ -bits alongside a tag  $t$  of length  $K$ -bits. The tag  $t$  should depend both on the message and on the private key.

The entire system should have polynomial complexity.

To implement this system, firstly we generated a random shared key  $k$  and a random message  $u$  that we want to transmit.

The `sign_message(u, k)` signing function was implemented in this way:

- We used the `generate_tag(u, k)` function to generate the tag. The function performs these steps:
  - Convert  $u$  and  $k$  from binary to base 10
  - Used our `sum_digits(num)` function to get the sum of the digits of the number
  - Computes the  $s = s_u \cdot s_k$  i.e the product of the sum of the digits of  $k$  and  $u$
  - Finally convert  $s$  into binary to obtain the tag
- After having computed the tag, we sign the message by appending the tag to the original message.

To verify the tag we had to make sure that the received message-tag pair was correct, i.e. there was no mismatch between the expected tag for the received message and the actual received tag.

To achieve this we implemented the `verify_tag(x, k)` that proceeded in this way:

- We divide  $x$  into message  $u_{received}$  and  $t_{received}$  based on the expected length of the message
- We use the `generate_tag(u, k)` with the received message and the shared symmetric key  $k$  to compute the correct tag  $t$
- We compare the received tag  $t_{received}$  with the computed tag  $t$
- If there is a mismatch we set the error bit  $b = 1$  and discard the message, otherwise we set  $b = 0$  and accept it as valid

We then evaluated the computational complexity of our scheme in this way:

- a. We fixed the key length  $K$

- b. We performed the signing and verification operations multiple times with the same parameters
- c. We computed the average signing and verification times
- d. We increased the length of the message  $M$  and repeated the transmission and verification steps

The figure below shows the results of this computation:

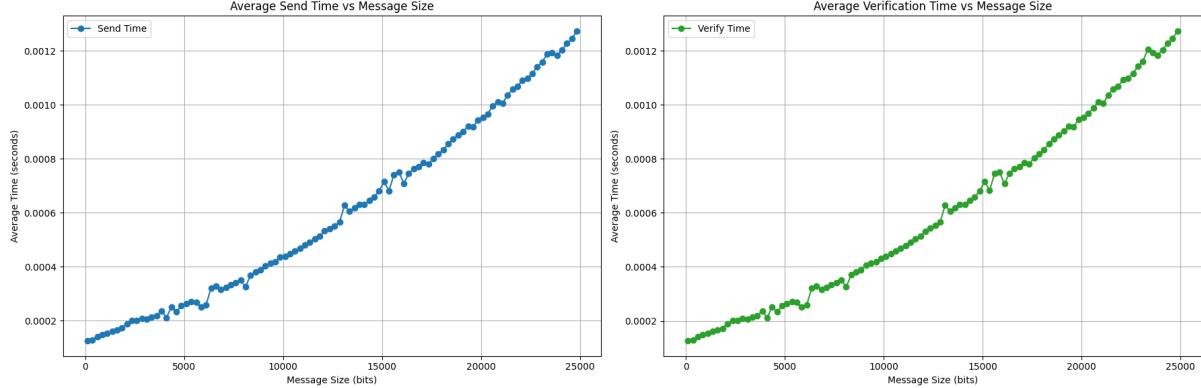


Figure 1: Variation of the send and verification times with an increasing key size

We repeated the same procedure, but instead fixed  $M$  and varied  $K$  and obtained the results shown below.

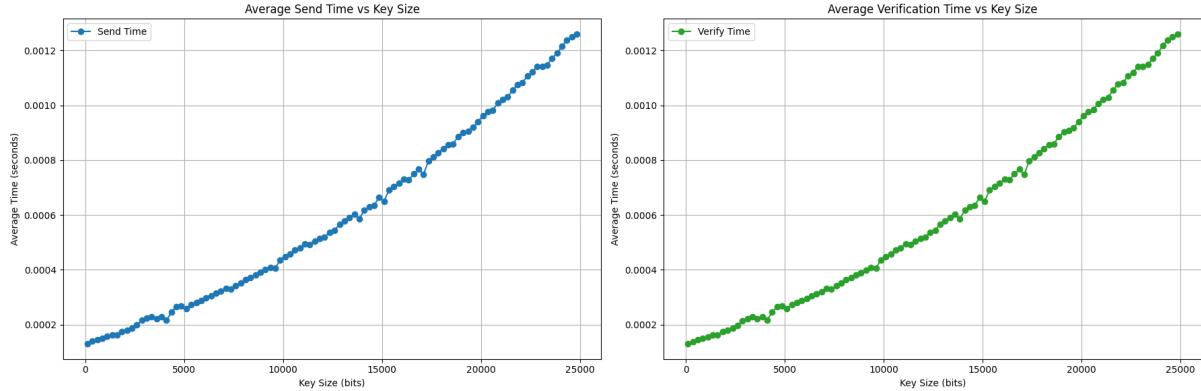


Figure 2: Variation of the send and verification times with an increasing message size

As expected, since both the tag creation and verification depend on the length of the key  $K$  and of the message  $M$ , we have a linear trend both for the implementation of  $S(k, u)$  and  $V(k, \tilde{x})$ .

## 2 Task 2 - Substitution attack

In this task, we had to implement a substitution attack.

In our scenario we assumed that the attacker had the capabilities to **intercept** and **block** the transmission of the legitimate message-tag pair  $x = (u, t)$ .

After having intercepted the legitimate transmission, the attacker would substitute it with a different one and transmit it to the receiver.

The aim of the attacker is to make it so that the transmitted message is accepted by the receiver ( $b = 0$ ).

To achieve this, we implemented the following procedure:

- Divide the received  $x$  into received message  $u\_received$  and received tag  $t\_received$

- Convert the received values into base 10.
- Perform the sum of the digit operation and obtain  $s$  from the tag and  $s_u$  from the message
- Since the tag is computed through  $s = s_u \cdot s_k$  and we know  $s_u$  from the received message, we can compute  $s_k = s/s_u$
- Finally, use  $s_k$  and the substitution message to compute a valid tag for the new message

This attack is very powerful, as it leverages a weakness in our authentication method. In fact, if the attacker has intercepted a valid message-tag pair, he can execute a substitution attack **with 100% probability of success**.

As before, we evaluated the execution time with increasing message and key size.

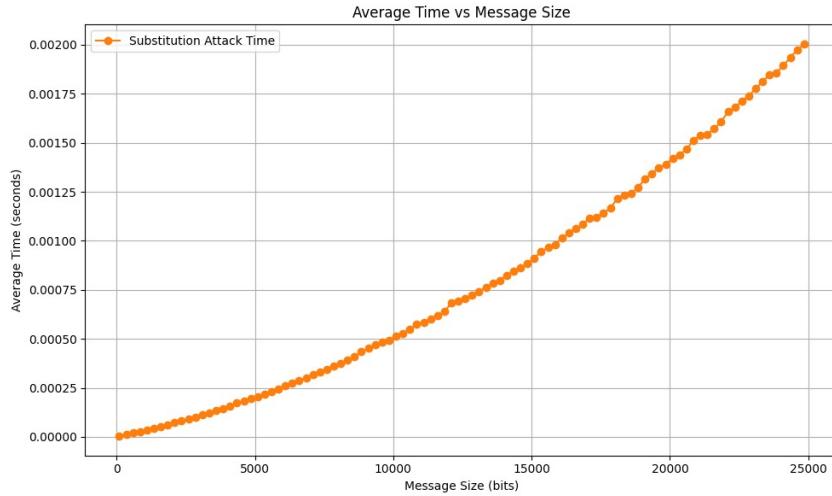


Figure 3: Variation of the average computational time required for a substitution attack as the message size increases

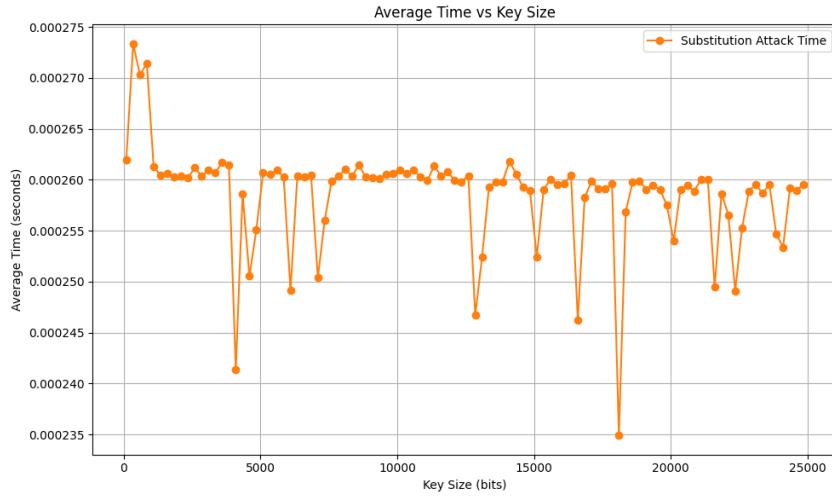


Figure 4: Variation of the average computational time required for a substitution attack as the key size increases

As expected, we saw a linear trend in the time complexity of the attack with an increasing message size. This is because of the various steps that the attacker needs to perform to create a valid tag, which depend on the length of the message. On the contrary, since the attacker does not require access to the key  $k$  to perform the attack, there is no evident correlation between key size and average attack time.

### 3 Task 3 - Design and implement a forging attack

In this task we were asked to implement a **forging attack**. Compared to Task 2, in this task, the attacker does not have the ability to intercept messages to extract the tag and instead has to rely on a *probabilistic approach*.

First, we needed to see if there was any pattern in the generation of the generation of the tags. Specifically, we implemented a `compute_all_digit_sums(N)` function, which, given a number of bits  $N$ , returns all the possible results of the base 10 digit sum that we use to generate the tag.

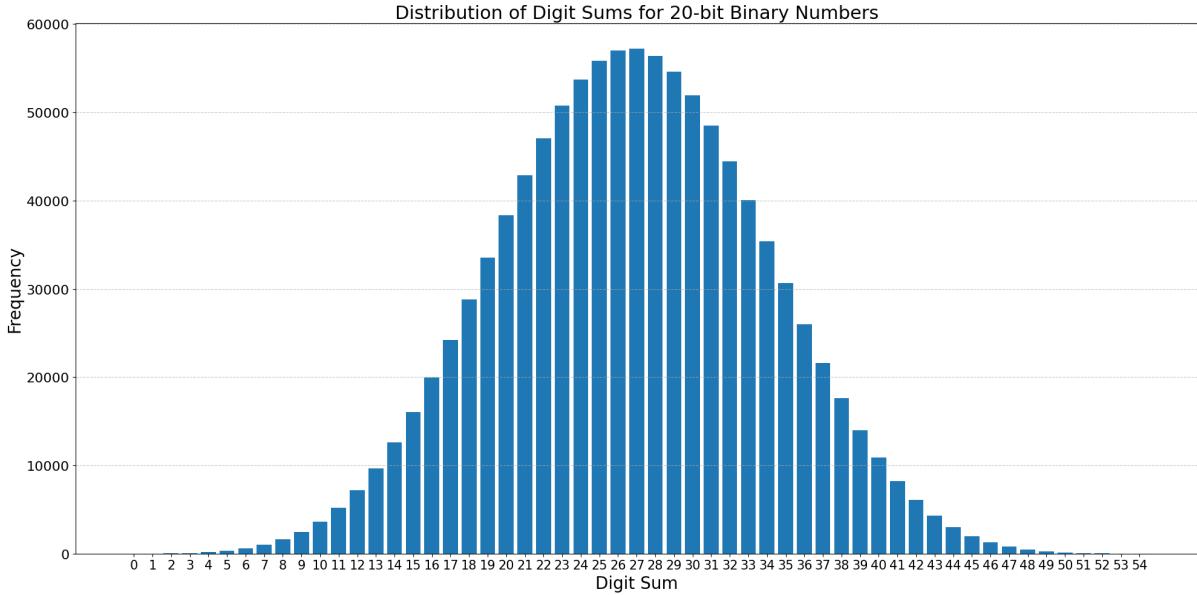


Figure 5: Distribution of possible sum of digits result with  $N = 20$  bits

As we can see, not all the outputs for the sum of the digits are equally probable. Thus, it might be useful to start from the most probable values when attempting our forging attack.

First, we needed a way to find the peak of the Gaussian distribution. To do this, we wrote the `find_maximum(K)` function that, given the size of the key, finds the value with the highest possible sum of digits with that number of bits. For example, the `find_maximum` function called with a key size of 9 would find that the highest possible sum of digits value would be produced by 499 which is the number under 512 that produces the highest digits sum.

Then, after knowing the maximum value, we computed its sum of digits and divided it by two to find the center point of the distribution. Specifically:

```
max_value = sum_digits(find_maximum(K))
most_likely_value = int(max_value / 2)
```

The `forging_attack(u_forged, K)` computes the most likely value with the technique shown above, crafts the correct tag with our forged message and transmits it. If the value is rejected, the function tries the neighboring values, starting from the two next to the most likely value and moving in both directions along the gaussian distribution.

#### 3.1 Computational Time Evaluation

We evaluated the average computational time in the same way as in Task 2, by varying key and message size and observing the effect on the attack time. In this case, taking the **average** is even more important than before, as the individual observations can vary a lot because of the stochastic nature of the attack. The results are shown below:

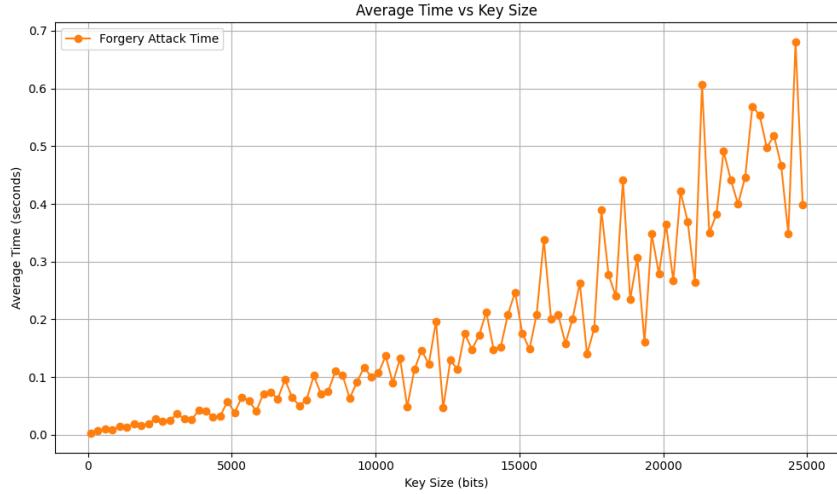


Figure 6: Variation of the average computational time required for a forging attack as the key size increases

As we can see, there is a correlation between key size and attack time, as a large key requires more attempts before the attack can succeed.

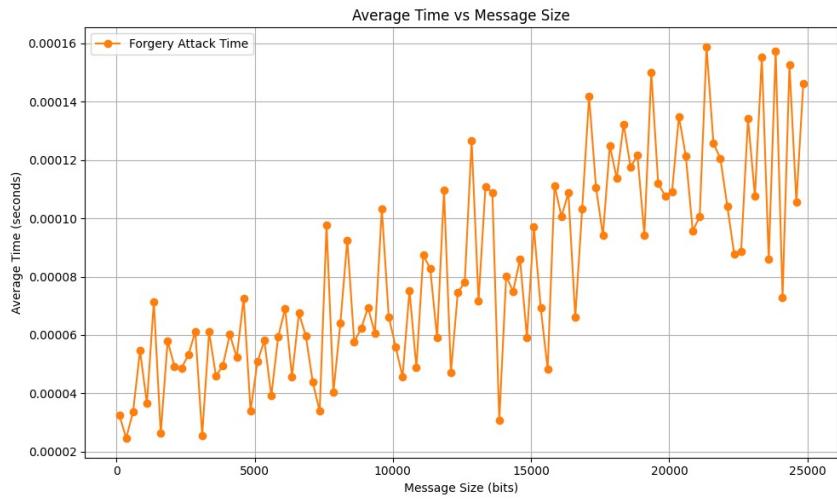


Figure 7: Variation of the average computational time required for a forging attack as the message size increases

There seems to also be a correlation between message size and attack time, tho this correlation is not as evident as with the key size.

### 3.2 Success probability

Compared with the substitution attack of Task 2, that had a deterministic success probability, this attack is not guaranteed to succeed on the first try, despite us trying the most likely tag with that key size.

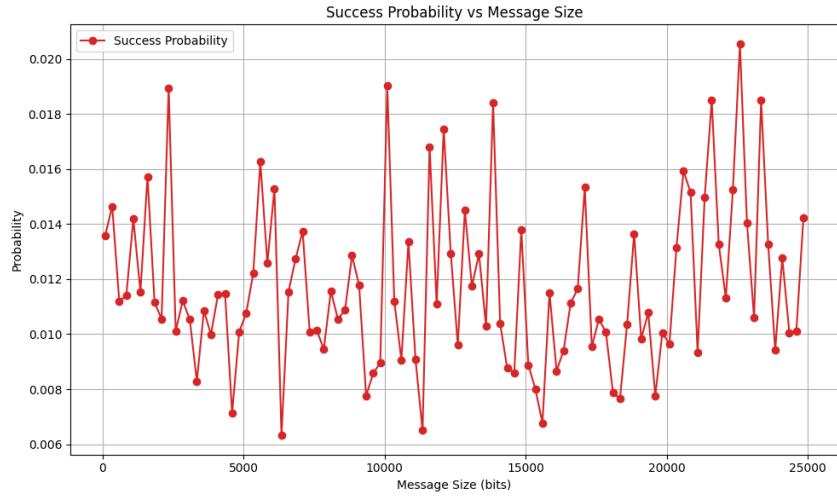


Figure 8: Variation of success probability for a forging attack as the message size increases

As we can see from the graph above, the success probability of the attack is not correlated with the message size, as it does not rely on any information provided by the message.

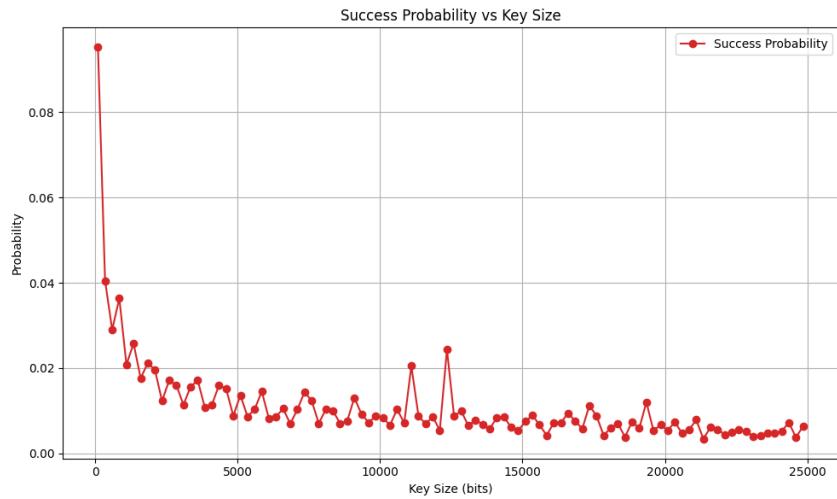


Figure 9: Variation of success probability for a forging attack as the key size increases

On the contrary, a longer key means lower probability of guessing the correct tag value as there are more possible options that the attacker needs to try. This is why we see a decreasing trend between success probability and key size.