Cairo University
Faculty of Engineering

Department of Computer Engineering

Cryptography Project

Submitted to

Dr. Samir Shaheen

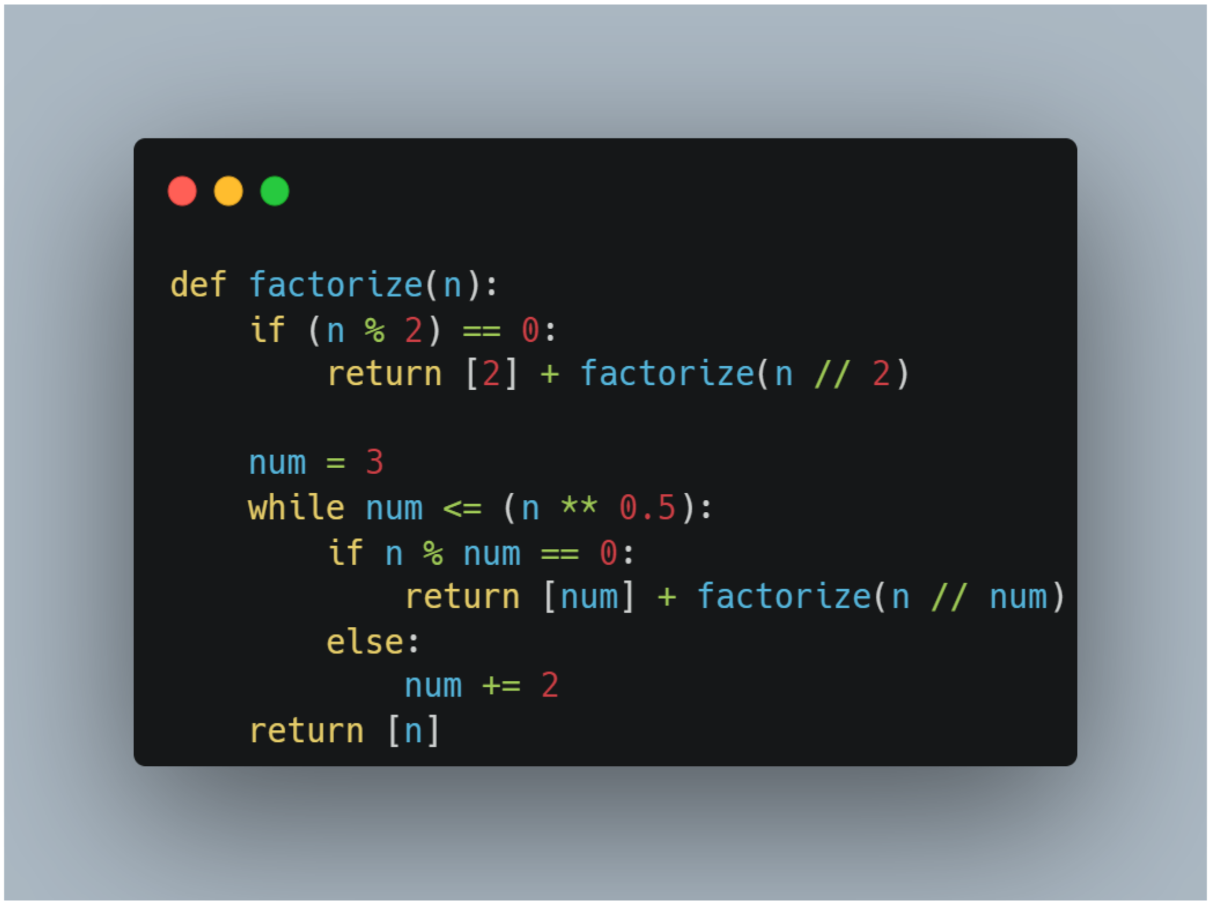
Eng. Khaled Moataz

Submitted by

Name	Section	B. N
Beshoy Morad Atya	1	20

RSA Brute-force Attack Test

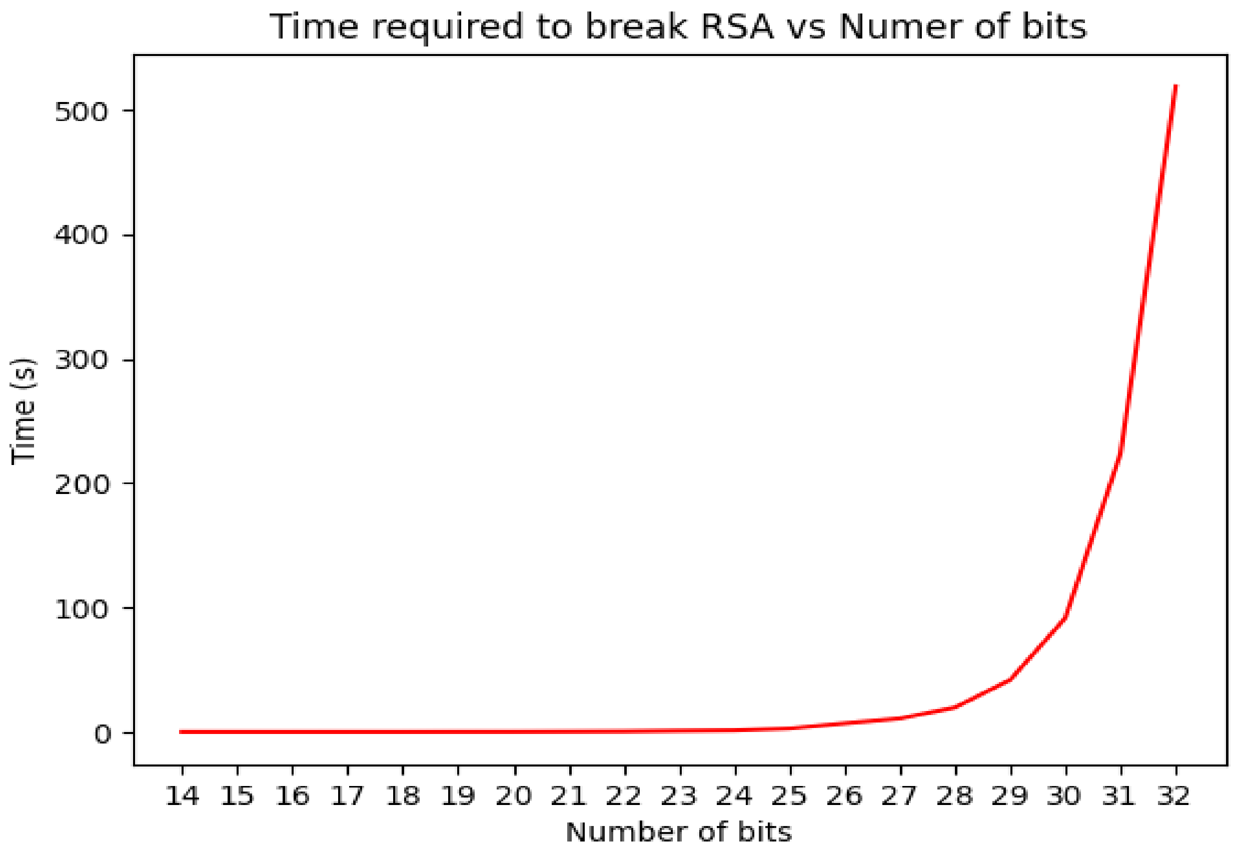
In the brute-force attack we try to factorize a set of public keys and measure the time in each case. We used a very straightforward algorithm for integer factorization that simply searches for divisors of the given integer by considering those odd ones lying only in the range up till the integer's square root. Once a divisor is found it's guaranteed to be prime (because we are iterating from small numbers to larger ones) and dividing the public key by that should yield the other prime.



```
def factorize(n):
    if (n % 2) == 0:
        return [2] + factorize(n // 2)

    num = 3
    while num <= (n ** 0.5):
        if n % num == 0:
            return [num] + factorize(n // num)
        else:
            num += 2
    return [n]
```

Analysis Results & Conclusions



In the mathematical brute-force attack. I started by generating a list of public keys ranging from [14 to 32 bits], then ran the factoring algorithm on the whole array and reported the time results in a text file.

Notice that although it seems that the time for breaking smaller is the same (the flat region on the left.), it's not actually flat but it's just that the scale on y is large due to the time taken to break the very large keys.

The trend is overall clearly exponential which is indeed the known time complexity of prime factorization. Notice that we were easily able to break RSA for 30 bits which means that unlike symmetric encryption we can't use such small key-lengths which justifies why we resorted to using 1024-bit RSA in our algorithm's implementation.