Assignment 1

Security and Privacy (CC2009)

Dante Rotnes (202105772)

Matheus Rodrigues (202101391)

Porto, Portugal
March 20th, 2024

1. Introduction

Many different cryptography algorithms have been developed in order to guarantee reliability and safety when ciphering data throughout the internet. In modern days, one can choose from a variety of different cryptographic mechanisms such as public-key cryptography (asymmetric), shared-key cryptography (symmetric) and hashing. Each of those count with a diversity of algorithms that allow a person to secure their data the best way possible. Among those algorithms, the RSA for asymmetric cryptography, AES for symmetric cryptography and SHA256 are some of the most widely utilized nowadays. This paper relates a performance benchmark of those cryptographic mechanisms, attesting its code implementation, its performance in different experimental environments and a comparison of the results encountered.

2. Code

All code was implemented in python. Each algorithm has its own python file and those have the same structure: algorithm implementation; testing; logging; and chart generation.

Algorithm implementation: All three algorithms RSA, AES and SHA256 were implemented following the documentation of the cryptography library [1].

Testing: The script conducts two types of experiments: random file encryption/decryption or hashing and same file encryption/decryption or hashing.

- Random File Experiment: For each specified file size, the script generates random data for each encryption operation (in the case of AES, also a unique nonce). It then measures the average encryption and decryption/hashing times over a predetermined number of repetitions. This experiment provides insights into the algorithms' performances across different file sizes and random data.
- Same File Experiment: In this experiment, the script encrypts and decrypts/hashes the same data over a fixed number of repetitions to evaluate the impact of repetition on performance. By repeatedly encrypting and decrypting/hashing the same data, it assesses the algorithms' efficiency in handling repetitive operations.

Logging: Detailed logging is integrated into the script, saving information about each encryption and decryption/hashing operation. This includes the file size, input data, encryption time, decryption time, hashing time, and average times.

Logging enables comprehensive analysis and evaluation of the algorithms performances under various conditions.

Chart generation: Upon completion of the experiments, the script generates visual representations of the results using matplotlib. It plots the average encryption and decryption/hashing times in microseconds against the file sizes in bytes for both random and same file experiments. These plots offer clear insights into the algorithms performances and help interpret the experimental outcomes.

```python
def rsa_encrypt(data, public_key):
    start_time = time.time()
    cipher_text = public_key.encrypt(
        data,
        padding.OAEP(
            mgf=padding.MGF1(algorithm=hashes.SHA256()),
            algorithm=hashes.SHA256(),
            label=None
        )
    )
    end_time = time.time()
    encryption_time = end_time - start_time
    return cipher_text, encryption_time


def rsa_decrypt(cipher_text, private_key):
    start_time = time.time()
    plain_text = private_key.decrypt(
        cipher_text,      You, 2 days ago • Update rsa.py
        padding.OAEP(
            mgf=padding.MGF1(algorithm=hashes.SHA256()),
            algorithm=hashes.SHA256(),
            label=None
        )
    )
    end_time = time.time()
    decryption_time = end_time - start_time
    return plain_text, decryption_time
```

RSA Encryption and Decryption functions

```python
def aes_encrypt(message: bytes, nonce: bytes, aesgcm) -> bytes:
    start_time = time.time()
    cipher_text = aesgcm.encrypt(nonce, message, None)
    end_time = time.time()
    encryption_time = end_time - start_time
    return cipher_text, encryption_time

def aes_decrypt(message: bytes, nonce: bytes, aesgcm) -> str:
    start_time = time.time()
    plain_text = aesgcm.decrypt(nonce, message, None)
    end_time = time.time()
    decryption_time = end_time - start_time
    return plain_text, decryption_time
```

AES Encryption and Decryption functions

```python
def sha256_hash(data, log_file):
    digest = hashes.Hash(hashes.SHA256())
    start_time = time.time()
    digest.update(data)
    hash_result = digest.finalize()
    end_time = time.time()
    hash_time = end_time - start_time
    return hash_result, hash_time
```

SHA256 Hashing function

```python
def generate_key_pair():
    private_key = rsa.generate_private_key(
        public_exponent=65537,
        key_size=2048,
        backend=default_backend()
    )
    public_key = private_key.public_key()
    return private_key, public_key
```

RSA Key Pair Generation

```python
# Generate key
key = AESGCM.generate_key(bit_length=256)
aesgcm = AESGCM(key)
```

AES Key Generation

```python
def run_enc_dec_random(file_sizes, repetitions, log_file, aesgcm):
    encryption_times = []
    decryption_times = []

    log_file.write(f"------------ Number of samples: {repetitions} ------------\n")

    for size in file_sizes:
        total_encryption_time = 0
        total_decryption_time = 0

        log_file.write(f"\n------------ Starting ------------\n")
        log_file.write(f"File size: {size} bytes\n")

        for i in range(repetitions):
            data = urandom(size)
            nonce = urandom(12)

            log_file.write("\nFile: " + str(i) + "\n")

            log_file.write("Data: "+str(data.hex()) + "\n")

            cipher_text, encryption_time = aes_encrypt(data, nonce, aesgcm)
            total_encryption_time += encryption_time
            log_file.write("Encryption: " + str(cipher_text.hex()) + "\n")
            log_file.write("Total encryption time: " + str(total_encryption_time) + "\n")

            plain_text, decryption_time = aes_decrypt(cipher_text, nonce, aesgcm)
            total_decryption_time += decryption_time
            log_file.write("Decryption: " + str(plain_text.hex())+ "\n")
            log_file.write("Total decryption time: " + str(total_decryption_time) + "\n")

        avg_encryption_time = total_encryption_time / repetitions
        avg_decryption_time = total_decryption_time / repetitions

        log_file.write(f"\n------------ Ending ------------\n")
        log_file.write(f"Average encryption time: {avg_encryption_time}\n")
        log_file.write(f"Average decryption time: {avg_decryption_time}\n\n")

        encryption_times.append(avg_encryption_time)
        decryption_times.append(avg_decryption_time)

    return encryption_times, decryption_times
```

AES Random File Test

The full code implementation can be found on github (https://github.com/dRotnes/security_privacy/tree/main/trabalho1).

3. Test results

As mentioned in the previous topic, the code was prepared to run two different kind of tests, one where many different files would be encrypted and decrypted/hashed only once and another where the same file would be encrypted and decrypted/hashed multiple times.

In order to generate statistically relevant results, each of the tests were made with 1000 repetitions for each algorithm. For the first test, for each file size specified, 1000 random files were generated and passed through the algorithms with only the algorithm encryption and decryption/hashing being timed. From that, the average time for each file size was calculated. For the second test, for each file size only one file was generated and passed through the algorithms 1000 times and the average time per file size was saved.

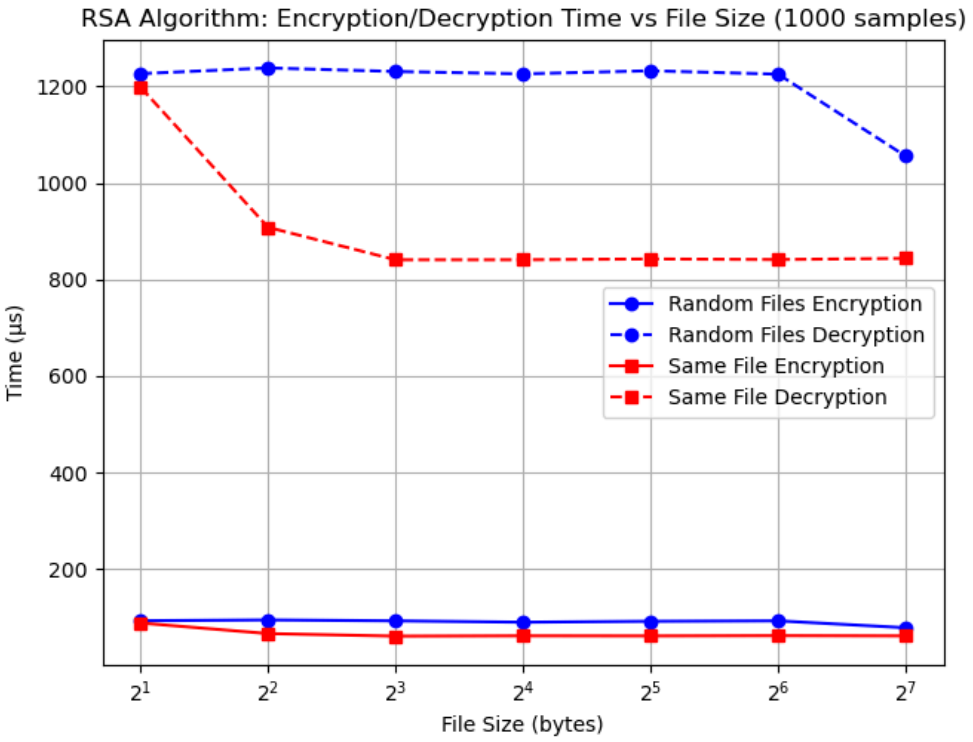
After running the tests for each algorithm, the charts were created to show the comparisons between both tests.

The following results were encountered:

- RSA file sizes: 2, 4, 8, 16, 32, 64 and 128 Bytes.
- AES file sizes: 8, 64, 512, 4096, 32768, 262144, and 2097152 Bytes.
- SHA256 file sizes: 8, 64, 512, 4096, 32768, 262144, and 2097152 Bytes.

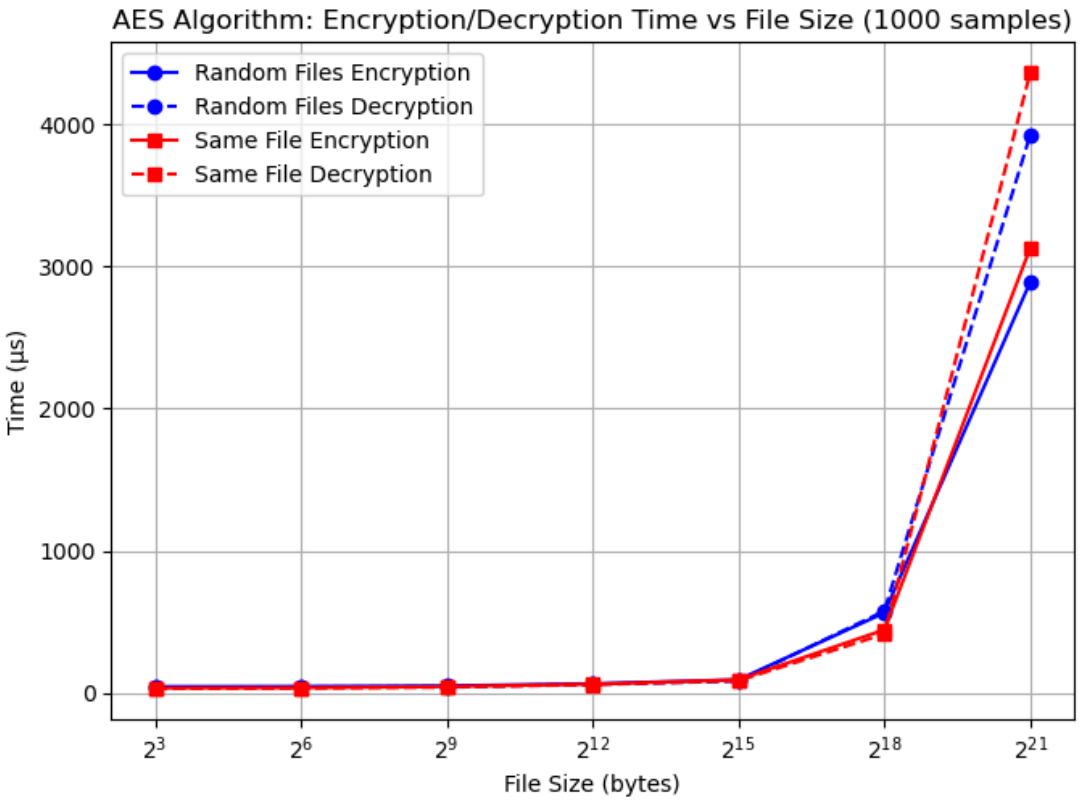
1st testing environment:

Operating System: Ubuntu 22.04.4 LTS 64-bit with GNOME 42.9
CPU: AMD Ryzen 3 3250U with Radeon Graphics
Memory: 8 GB DDR4 RAM
Graphics: AMD® Radeon vega 3 graphics
Disk Capacity: 256 GB



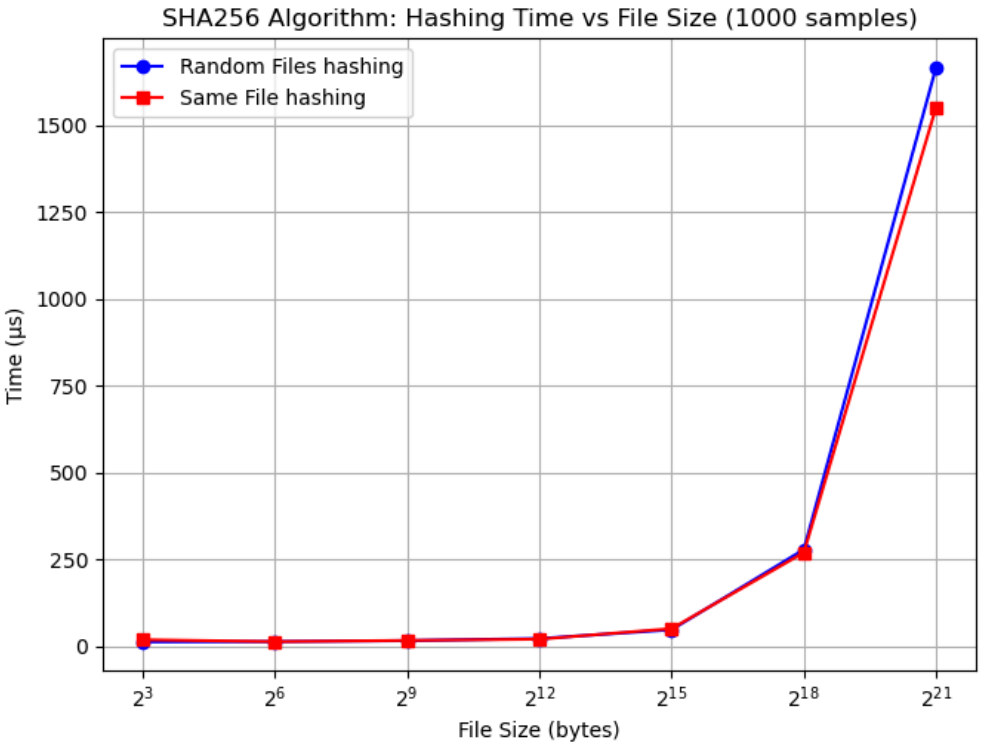

```
Algorithm: RSA
Generating plot and log file...

------------------------------------

------------ Random Files Test ------------

File size: 2 | Average encryption time: 9.283900260925293e-05 | Average decryption time: 0.0012266173362731933

File size: 4 | Average encryption time: 9.443688392639l6e-05 | Average decryption time: 0.0012384028434753419

File size: 8 | Average encryption time: 9.261178970336914e-05 | Average decryption time: 0.0012310636043548584

File size: 16 | Average encryption time: 8.975934982299805e-05 | Average decryption time: 0.0012261579036712646

File size: 32 | Average encryption time: 9.166574478149414e-05 | Average decryption time: 0.001232541799545288

File size: 64 | Average encryption time: 9.260344505310058e-05 | Average decryption time: 0.0012255606651306152

File size: 128 | Average encryption time: 7.831931114196777e-05 | Average decryption time: 0.0010557825565338135

------------ Same Files Test ------------

File size: 2 | Average encryption time: 8.814620971679687e-05 | Average decryption time: 0.0011987509727478028

File size: 4 | Average encryption time: 6.617689132690429e-05 | Average decryption time: 0.0009081356525421142

File size: 8 | Average encryption time: 6.09276294708252e-05 | Average decryption time: 0.0008408520221710206

File size: 16 | Average encryption time: 6.156396865844727e-05 | Average decryption time: 0.0008409321308135986

File size: 32 | Average encryption time: 6.136894226074218e-05 | Average decryption time: 0.0008425486087799073

File size: 64 | Average encryption time: 6.187224388122559e-05 | Average decryption time: 0.0008413527011871338

File size: 128 | Average encryption time: 6.144142150878906e-05 | Average decryption time: 0.0008438327312469483
```

RSA Algorithm average encryption/decryption times (1st testing environment)




```
Algorithm: AES
Generating plot and log file...

-----------------------------------

------------ Random Files Test ------------

File size: 8 | Average encryption time: 4.514312744140625e-05 | Average decryption time: 3.994154930114746e-05

File size: 64 | Average encryption time: 4.647183418273926e-05 | Average decryption time: 4.2269229888916014e-05

File size: 512 | Average encryption time: 5.016446113586426e-05 | Average decryption time: 4.4541358947753905e-05

File size: 4096 | Average encryption time: 6.541800498962402e-05 | Average decryption time: 6.060481071472168e-05

File size: 32768 | Average encryption time: 9.325337409973144e-05 | Average decryption time: 8.385825157165527e-05

File size: 262144 | Average encryption time: 0.0005633509159088134 | Average decryption time: 0.0005765054225921631

File size: 2097152 | Average encryption time: 0.0028877451419830323 | Average decryption time: 0.0039221239089965824

------------ Same Files Test ------------

File size: 8 | Average encryption time: 3.457999229431152e-05 | Average decryption time: 3.129792213439941e-05

File size: 64 | Average encryption time: 3.632783889770508e-05 | Average decryption time: 3.318977355957031e-05

File size: 512 | Average encryption time: 4.5238494873046875e-05 | Average decryption time: 4.034829139709473e-05

File size: 4096 | Average encryption time: 6.242537498474122e-05 | Average decryption time: 5.8908462524414065e-05

File size: 32768 | Average encryption time: 9.315371513366699e-05 | Average decryption time: 8.566832542419434e-05

File size: 262144 | Average encryption time: 0.0004442367553710937 | Average decryption time: 0.0004148898124694824

File size: 2097152 | Average encryption time: 0.003126721143722534 | Average decryption time: 0.004357632160186768
```

AES Algorithm average encryption/decryption times (1st testing environment)

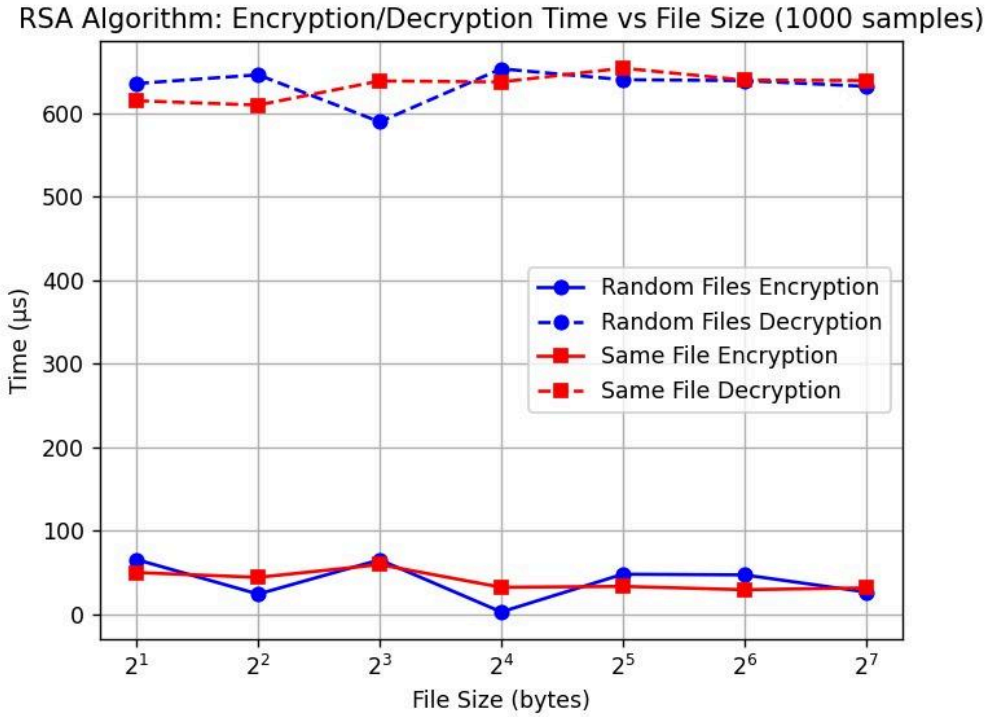



```
Algorithm: SHA256
Generating plot and log file...

-----------------------------------

------------ Random Files Test ------------

File size: 8 | Average hashing time: 1.2504100799560546e-05

File size: 64 | Average hashing time: 1.3602018356323243e-05

File size: 512 | Average hashing time: 1.6225814819335937e-05

File size: 4096 | Average hashing time: 2.225065231323242e-05

File size: 32768 | Average hashing time: 4.7639131546020505e-05

File size: 262144 | Average hashing time: 0.00027855539321899414

File size: 2097152 | Average hashing time: 0.0016661670207977295

------------ Same Files Test ------------

File size: 8 | Average hashing time: 1.882743835449219e-05

File size: 64 | Average hashing time: 1.3367891311645508e-05

File size: 512 | Average hashing time: 1.678943634033203e-05

File size: 4096 | Average hashing time: 2.0503759384155272e-05

File size: 32768 | Average hashing time: 5.072855949401855e-05

File size: 262144 | Average hashing time: 0.0002696683406829834

File size: 2097152 | Average hashing time: 0.0015513110160827637
```

SHA256 Algorithm average hashing times (1st testing environment)

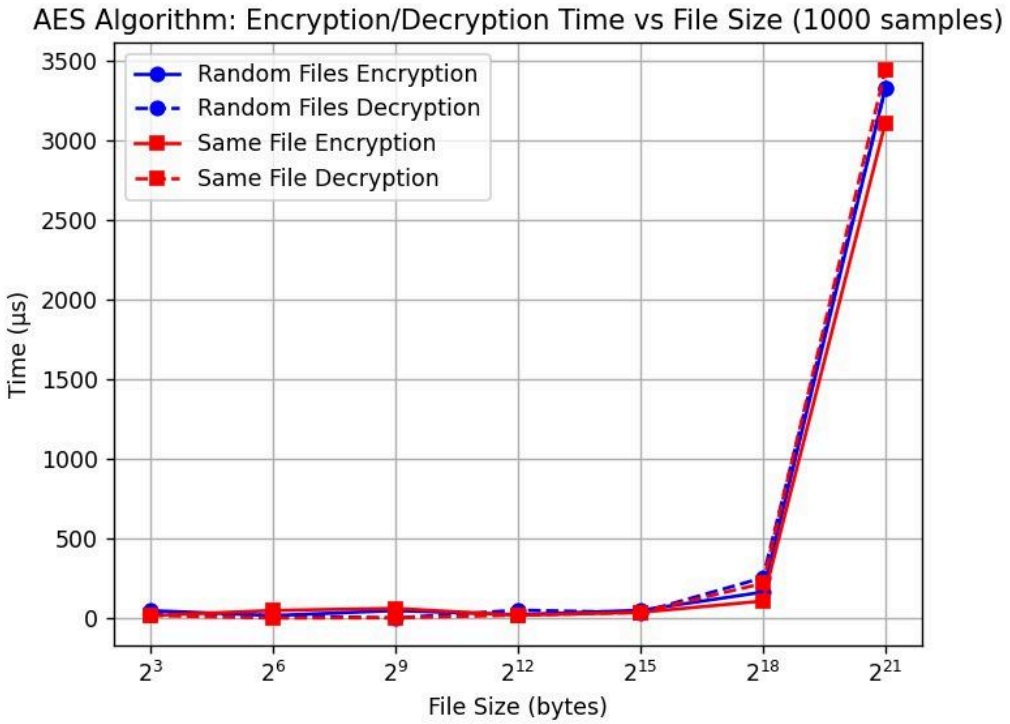
2nd testing environment:

Operating System: Windows 11
CPU: Intel(R) Core(TM) i7-7500U @ 2.70GHz 2.90 GHz
Memory: 24 GB DDR4 RAM
Graphics: Nvidia 940mx 4gb
Disk Capacity: 256 GB



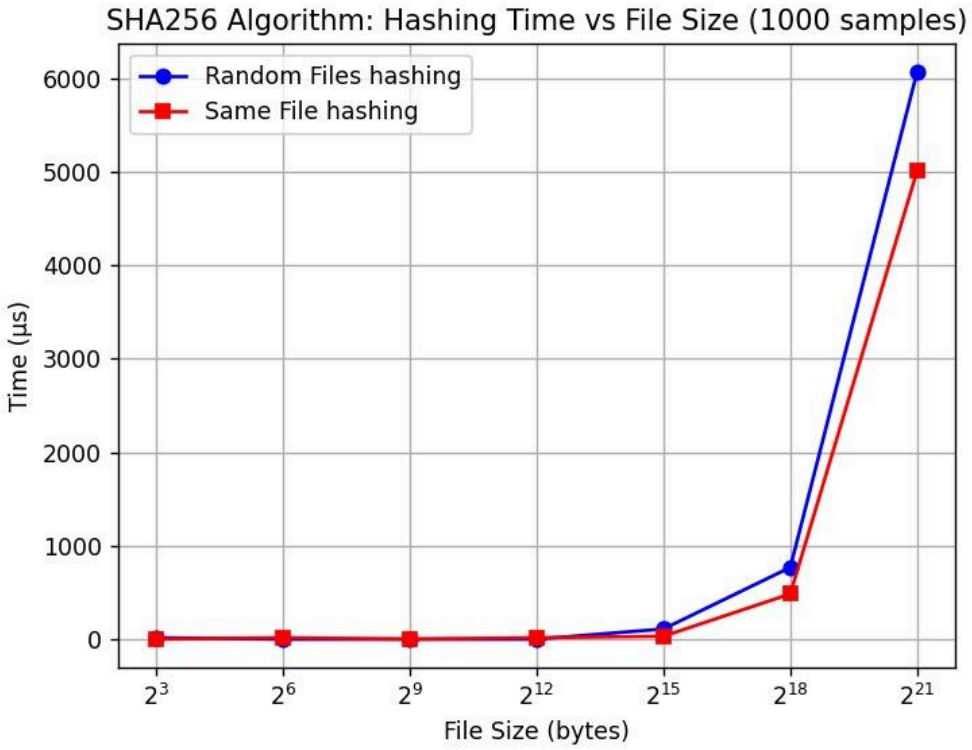

```
Algorithm: RSA
Generating plot and log file...

------------------------------------

------------ Random Files Test ------------

File size: 2 | Average encryption time: 6.545281410217285e-05 | Average decryption time: 0.0006357171535491944

File size: 4 | Average encryption time: 2.4057626724243164e-05 | Average decryption time: 0.0006461868286132813

File size: 8 | Average encryption time: 6.514215469360352e-05 | Average decryption time: 0.0005898063182830811

File size: 16 | Average encryption time: 2.6538372039794922e-06 | Average decryption time: 0.0006533310413360595

File size: 32 | Average encryption time: 4.78978157043457e-05 | Average decryption time: 0.0006402795314788818

File size: 64 | Average encryption time: 4.703974723815918e-05 | Average decryption time: 0.0006392190456390381

File size: 128 | Average encryption time: 2.6542186737060545e-05 | Average decryption time: 0.0006323108673095703

------------ Same Files Test ------------

File size: 2 | Average encryption time: 4.988551139831543e-05 | Average decryption time: 0.0006150884628295898

File size: 4 | Average encryption time: 4.406118392944336e-05 | Average decryption time: 0.0006098954677581787

File size: 8 | Average encryption time: 5.934476852416992e-05 | Average decryption time: 0.0006388020515441894

File size: 16 | Average encryption time: 3.220486640930176e-05 | Average decryption time: 0.0006376287937164307

File size: 32 | Average encryption time: 3.328442573547363e-05 | Average decryption time: 0.0006541283130645752

File size: 64 | Average encryption time: 2.9264211654663085e-05 | Average decryption time: 0.0006398565769195557

File size: 128 | Average encryption time: 3.14784049987793e-05 | Average decryption time: 0.0006392960548400879
```

RSA Algorithm average encryption/decryption times (2nd testing environment)




```
Algorithm: AES
Generating plot and log file...

------------------------------------

------------ Random Files Test ------------

File size: 8 | Average encryption time: 4.686141014099121e-05 | Average decryption time: 3.1271696090698245e-05

File size: 64 | Average encryption time: 1.5621423721313477e-05 | Average decryption time: 1.5619993209838867e-05

File size: 512 | Average encryption time: 4.686379432678223e-05 | Average decryption time: 0.0

File size: 4096 | Average encryption time: 2.1151542663574217e-05 | Average decryption time: 4.99913692474365246-05

File size: 32768 | Average encryption time: 4.88886833190918e-05 | Average decryption time: 3.4268856048583986e-05

File size: 262144 | Average encryption time: 0.00016274189949035646 | Average decryption time: 0.00025277495384216307

File size: 2097152 | Average encryption time: 0.003323958158493042 | Average decryption time: 0.0033211944103240967

------------ Same Files Test ------------

File size: 8 | Average encryption time: 1.5621185302734374e-05 | Average decryption time: 1.562190055847168e-05

File size: 64 | Average encryption time: 4.789638519287109e-05 | Average decryption time: 0.0

File size: 512 | Average encryption time: 6.0731172561645506e-05 | Average decryption time: 1.862049102783203e-06

File size: 4096 | Average encryption time: 1.7559289932250977e-05 | Average decryption time: 1.7645835876464844e-05

File size: 32768 | Average encryption time: 3.5487890243530275e-05 | Average decryption time: 3.5234928131103516e-05

File size: 262144 | Average encryption time: 0.00010690712928771973 | Average decryption time: 0.00021747279167175294

File size: 2097152 | Average encryption time: 0.003104910135269165 | Average decryption time: 0.003439479351043701
```

AES Algorithm average encryption/decryption times (2nd testing environment)




```
Algorithm: SHA256
Generating plot and log file...

-------------------------------------

------------ Random Files Test ------------

File size: 8 | Average hashing time: 1.5621423721313477e-05

File size: 64 | Average hashing time: 0.0

File size: 512 | Average hashing time: 0.0

File size: 4096 | Average hashing time: 0.0

File size: 32768 | Average hashing time: 0.00010928630828857422

File size: 262144 | Average hashing time: 0.0007679178714752197

File size: 2097152 | Average hashing time: 0.006065267801284 79

------------ Same Files Test ------------

File size: 8 | Average hashing time: 0.0

File size: 64 | Average hashing time: 1.5621185302734374e-05

File size: 512 | Average hashing time: 0.0

File size: 4096 | Average hashing time: 1.5622854232788086e-05

File size: 32768 | Average hashing time: 3.1205415725708006e-05

File size: 262144 | Average hashing time: 0.00048429274559020995

File size: 2097152 | Average hashing time: 0.005006857872009277
```

SHA256 Algorithm average hashing times (2nd testing environment)

3. Results analysis and comparisons

From the results gathered during both tests in the two different testing environments, we can clearly see how the three algorithms work in regards to different file sizes. For the RSA algorithm, in both testing environments, encryption time was significantly larger than the decryption time for all file sizes ranging from 2B to 128B, but both times were extremely similar regardless of the file size. This time difference is due to the different exponents utilized for the encryption and decryption. The public exponent is much smaller than the private exponent so the computation of raising the message to this small power is much quicker than raising it to the large private exponent.

Although in the first environment it's clear there is a difference in the decryption between the random files test and the same file test, this difference can be regarded to the environment the tests were ran in opposed to the algorithm itself

As for the AES algorithm, in both tests and environments the algorithm performed both encryption and decryption almost instantly for files with 8B (2^3) to 32768B (2^{15}) and saw a significant increase in encrypting and decrypting the subsequent sizes, with a exponential growth in time taken for files of 2097152B (2^{21}). The little difference in encryption and decryption times and the exponential growth in time taken for both operations is due to the logic of the algorithm. AES-256 (which means it utilizes a 256 bit key), is a symmetric cryptography algorithm, so both the encryption and decryption utilize the same key to generate the cipher and plain texts, meaning that the computational complexity of encryption and decryption operations is similar. At the same time, it works with rounds of sequential ciphering/decrypting of blocks of 128 bits, which explains the increase in time consumed when the file size is increased, as the larger files take more rounds to encrypt/decrypt.

At last, the results for the SHA256 algorithm show that it performs extremely well for files up to 4896B (2^{12}) and starts increasing the time taken to hash the input for the subsequent sizes, with an exponential growth towards the bigger files. Although the results show a slight difference in the time taken to hash random files and the same file of 2097152B (2^{21}), this can be due to hardware specifications rather than the algorithm performance itself. The rapid growth in time taken to hash the inputs as the file sizes grow beyond 4896B (2^{12}) can be regarded to the fact that SHA256 operates on fixed-size blocks of data of 512 bits and uses a series of simple bitwise operations and additions. With the exponential increase in file sizes, the number of blocks of data also increases, just like observed in the AES algorithm.

As for comparisons between different algorithms, it's visible that the AES encryption is slightly faster than the RSA encryption. Although with these tests it's harder to make such comparisons, as the file sizes tested are not the same, AES algorithm is optimized for larger files and we can see that the time consumed for RSA to encrypt a 2B (2^1) file is almost the same as for AES to encrypt a significantly larger file of 512B (2^9). At the same time, AES encryption and SHA256 hashing have a very similar outcome with the file sizes tested. While in the 1st environment of testing, SHA256 hashing and AES encryption times were almost the same, with differences of less than 0.002 seconds between both tests for files of 2097152B (2^{21}), in the 2nd environment AES encryption outperformed SHA256 hashing being almost twice as fast with times of ~ 0.0033s and ~ 0.0032s for random files and same file encryption while SHA256 had times of ~ 0.006s and ~ 0.005s.

4. Conclusion

As more and more cryptography algorithms are developed, it's hard to understand exactly what difference it makes utilizing one algorithm or another for the most common projects. While that may be true, it's important to understand the purpose of each algorithm and its performance in different aspects and environments. While RSA may take longer than AES to encrypt certain files, AES is a symmetric cryptography algorithm while RSA is a asymmetric cryptography algorithm, so both have different purposes and possibilities. At the same time, SHA256 comes in handy for hashing large files in little time, but as a hashing function, does not allow for encrypted data to be decrypted like both other algorithms. With that, choosing a cryptography algorithm nowadays has to do more with the intentions of a project than with the differences in performance between these algorithms.

5. References

[1] cryptography.io. (n.d.). Cryptography.io: Cryptography and secure communication library in Python. Retrieved [03/2024], from https://cryptography.io/en/latest/

[2] Matplotlib. (n.d.). Matplotlib: Visualization with Python. Retrieved [03/2024], from https://matplotlib.org/

[3] Kiteworks. (n.d.). AES-256 encryption. Retrieved [03/2024], from https://www.kiteworks.com/risk-compliance-glossary/aes-256-encryption/

[4] Encryption Consulting. (n.d.). What is RSA? Retrieved [03/2024], from https://www.encryptionconsulting.com/education-center/what-is-rsa/

[5] upGrad. (n.d.). SHA-256 Algorithm. Retrieved [03/2024], from https://www.upgrad.com/blog/sha-256-algorithm/