Vongai Kwenda

Project Milestone 4

Dataset Number: 37

25 November 2024

Milestone 4 Report

**Introduction**

This project entailed finding a secret message using my knowledge on symmetric and asymmetric encryption methods, namely AES, RSA Public and Private keys, and md5 hashes to determine the contents of a secret message from the data set I was given. The given dataset was a zip folder containing messages, AES keys, public-key/private-key pairs, a plain text message hash, and a plaintext AES hash.

**Environment**

My project was developed using the following software stack:

* **Programming languages:**
  + Python 3.x: I used the python programming language to implement the cryptographic description logic and elements and for the structure of my source code.
  + Hashlib (python module): I used this module to compute the MD5 hashes for the AES key and the decrypted messages hash to identify the secret message.
* **IDE:**
  + The integrated development environment (IDE) that I utilized for writing my source code, debugging, and using Python libraries was PyCharm. The development was streamlined by the console, and the error-highlighting inspector function.
* **Third-party libraries:**
  + PyCryptodome: As with a previous milestone, I used this python library for cryptographic operations. Namely:
    - PKCS1\_OAEP: For RSA decryption of the AES key
    - AES (CBC mode): For decrypting the encrypted messages using AES encryption. The IV was provided by the professor.
* **Operating system:** Windows 10 was the development environment where the Python code and tools were executed.

**Methodology**

**Design Approach:** My design approach for solving this problem involved replicating and building on the logic from the previous milestones (1, 2, and 3). I started by mapping out the structure of the problem and identifying key components in the decryption process. This helped me break the task down into manageable parts:

* **RSA Decryption:** The first layer of decryption was focused on decrypting the AES key using RSA, which was used to unlock the encrypted messages. Comparing the AES key hashes to the given hash identified the right AES key.
* **AES Decryption:** The second layer involved using the decrypted AES key to unlock the encrypted messages.
* **Message Verification:** The final step was comparing the decrypted messages’ MD5 hashes to the known hash to identify the correct message.

To ensure I understood the flow of the decryption process, I created a diagram in my notebook. This included the abstraction of the encryption layers, with **RSA on top of AES on top of the message**. I also sketched a simple flow chart that depicted the order of operations**: RSA -> AES -> Message**. This helped me visualize how each decryption step built upon the previous one.

I divided the problem-solving approach into two key parts:

* **Decrypting the AES key**: This was the first critical step, requiring RSA decryption for the AES key used to decrypt the actual messages.
* **Decrypting the messages**: Once the AES key was retrieved, I applied it to decrypt the messages.

**Division of labor:** Not applicable as I worked alone.

**Algorithm (Execution Path)**

My solution follows this logical path:

1. **Load RSA Private Key**: Load and process RSA private keys (stored in .pem files).
2. **Decrypt AES Key**: Use the RSA private key to decrypt the AES key.
3. **Verify AES Key**: After decryption, verify that the AES key matches the expected hash.
4. **Decrypt Messages**: With the correct AES key, decrypt each message file (.emsg).
5. **Verify Decrypted Message**: Compute the MD5 hash of each decrypted message and compare it against the provided master message hash (plain\_master\_message\_hash.md5).
6. **Identify Secret Message**: When a hash match is found, the correct message is identified as the secret message.

**My pseudocode**:

|  |
| --- |
| **For** each RSA private key:  Iterate over RSA private key files  **For** each AES key:  Decrypt AES key using RSA  If AES key hash matches:  Break out of the RSA key loop  **For** each encrypted message:  Decrypt message using AES key  Hash decrypted message  If hash matches master message hash:  Print the secret message  Exit |

**Challenges and solutions**

I had an infinite loop scenario following the identification of the correct RSA private key because of a complex nested loop of 3 loops structure. I solved this by breaking down the solution to have a nested loop of 2 for the first part (RSA to AES) and then a single loop for decrypting the final message and comparing it to the hash. Other challenges included:

1. **Debugging Decryption Issues**: Early on, the decrypted messages were coming out as gibberish or unreadable text. This was due to issues with padding during AES decryption.

*Solution*: I investigated the problem by troubleshooting and experimenting with different decryption methods (e.g., checking padding, verifying block sizes) until the messages were correctly decrypted.

1. **AI Chatbot Assistance**: When search engine results were not helpful, I turned to an AI chatbot for guidance on cryptography and troubleshooting issues related to RSA and AES decryption.

*Solution*: The AI chatbot helped clarify some concepts and provided useful insights into resolving decryption errors.

**Metrics**

**Content of dataset:** There were 90,001 messages, 200 symmetric AES keys, and 400 asymmetric RSA key pairs, and 2 hash files in the dataset.

**Execution time lapse:** The solution took 16 minutes, 19 seconds, 56 splits of a second to identify the secret message.

**How would the execution time be affected if the number of messages and keys were doubled?**

* If the number of messages and keys were doubled, the execution time would roughly double due to the exhaustive search process, assuming no optimization or parallel processing is used.

**Secret message:** why do kamikaze pilots wore helmets!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

# Bibliography

*AES Encryption and Decryption in Python [64, 128, 256]*. n.d. <https://codeigo.com/python/aes-encryption-and-decryption-in-python-64-128-256/>.

Basile. *AES Encryption & Decryption In Python: Implementation, Modes & Key Management*. 13 April 2022. <https://onboardbase.com/blog/aes-encryption-decryption/>.

*Finding Md5 of Files Recursively in Directory in Python*. 18 April 2024. <https://www.geeksforgeeks.org/finding-md5-of-files-recursively-in-directory-in-python/>.

*Is the original IV mandatory to decrypt an encrypted message?* 24 August 2020. <https://security.stackexchange.com/questions/237489/is-the-original-iv-mandatory-to-decrypt-an-encrypted-message>.

*RSA Encryption / Decryption - Examples in Python*. n.d. <https://wizardforcel.gitbooks.io/practical-cryptography-for-developers-book/content/asymmetric-key-ciphers/rsa-encrypt-decrypt-examples.html>.