Cryptography Project

RSA, AES & SALTED HASH

Professor SK MD Mizanur Rahman

ICET, Centennial College

December 13, 2023

Group 5

SYED - 301318212

DEEPAK BALAJI PRABU - 301304684

NIROJAN JEYANDRAN – 300807665

MOHAMMAD YOUSEF - 301335063

Abstract

This report scrutinizes the implementation of the Rivest–Shamir–Adleman (RSA) cryptosystem, Advanced Encryption Standard (AES) algorithm, and Salted Hash functions, emphasizing their roles in ensuring secure communication and data protection. The RSA section highlights key generation, encryption, and decryption processes, while the AES segment focuses on symmetric-key encryption. The Salted Hash functions elucidate secure password hashing. Identified limitations, such as random parameter selection in RSA, absence of non-prime number validation, and inadequate padding in AES, underscore potential vulnerabilities. Proposed solutions involve user-input flexibility, prime number validation, and key length standardization. These enhancements aim to fortify the security and usability of the cryptographic implementations.

Contents

List of Figures

1. Introduction

In the realm of secure communication and data integrity, the RSA cryptosystem, AES algorithm, and Salted Hash functions play pivotal roles. RSA leverages the mathematical complexity of prime numbers, while AES ensures robust symmetric-key encryption. Salted Hash functions bolster password security. This report delves into the intricacies of these cryptographic mechanisms, shedding light on their implementation details and security measures. However, identified limitations necessitate refinements, addressing issues like parameter randomness, prime number validation, and key length standardization, to fortify the overall security posture.

2. Rivest, Shamir, Adleman (RSA) Algorithm

RSA is a widely used public-key cryptosystem for secure communication and digital signatures. RSA is based on the mathematical properties of large prime numbers and their difficulty in factoring the product of two primes.

The code implements the RSA (Rivest–Shamir–Adleman) cryptosystem for public-key encryption. It generates a key pair, and encrypts and decrypts messages. Key generation involves selecting two prime numbers, calculating their product (n), and finding a public exponent (e) with its corresponding private exponent (d). The 'Encrypt' function converts a plaintext message into an array of integers using modular exponentiation, and 'Decrypt' reverses the process. The code ensures that the selected primes are valid, handles edge cases, and employs modular exponentiation for efficient computations, adhering to RSA's mathematical principles for secure communication.

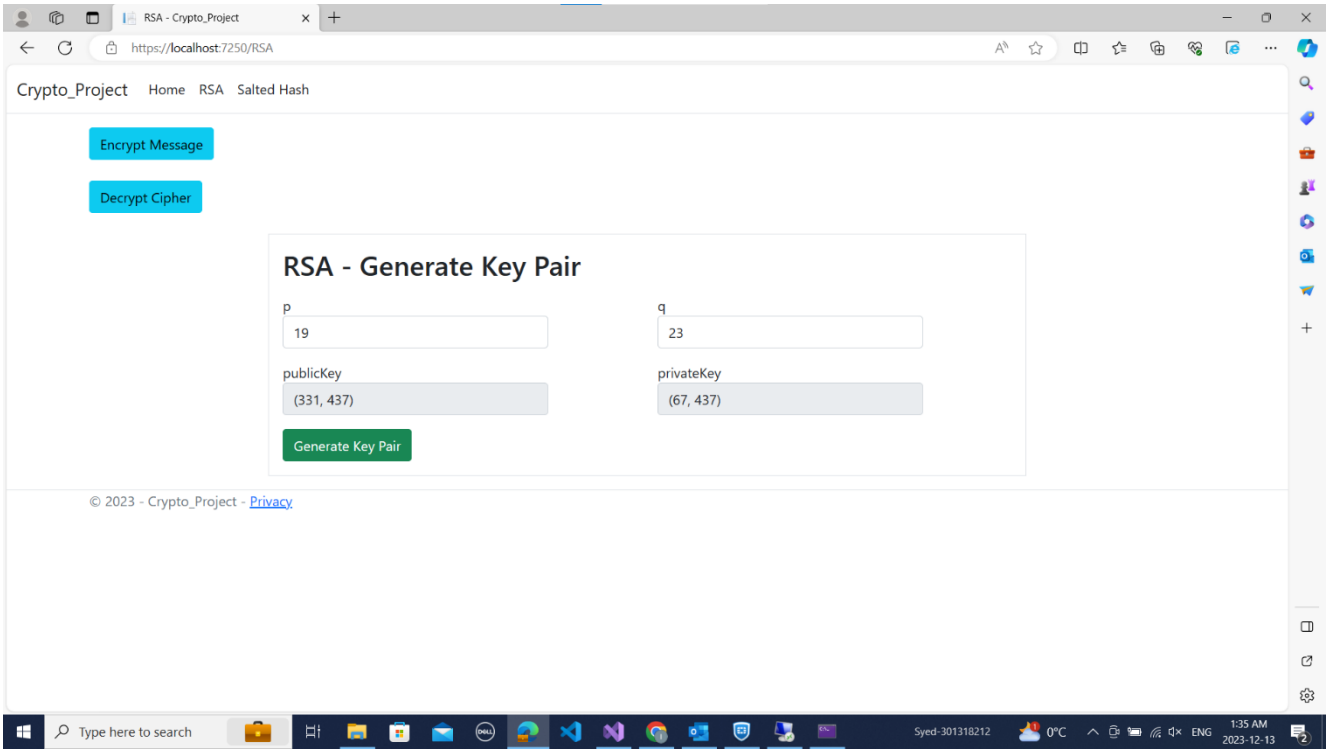

Figure 1 - Generating Keys for RSA

A sender uses the recipient's public key to encrypt a message. can be freely shared, allowing others to encrypt messages for the key's owner.

Convert from message to Cipher text (Encrypt)

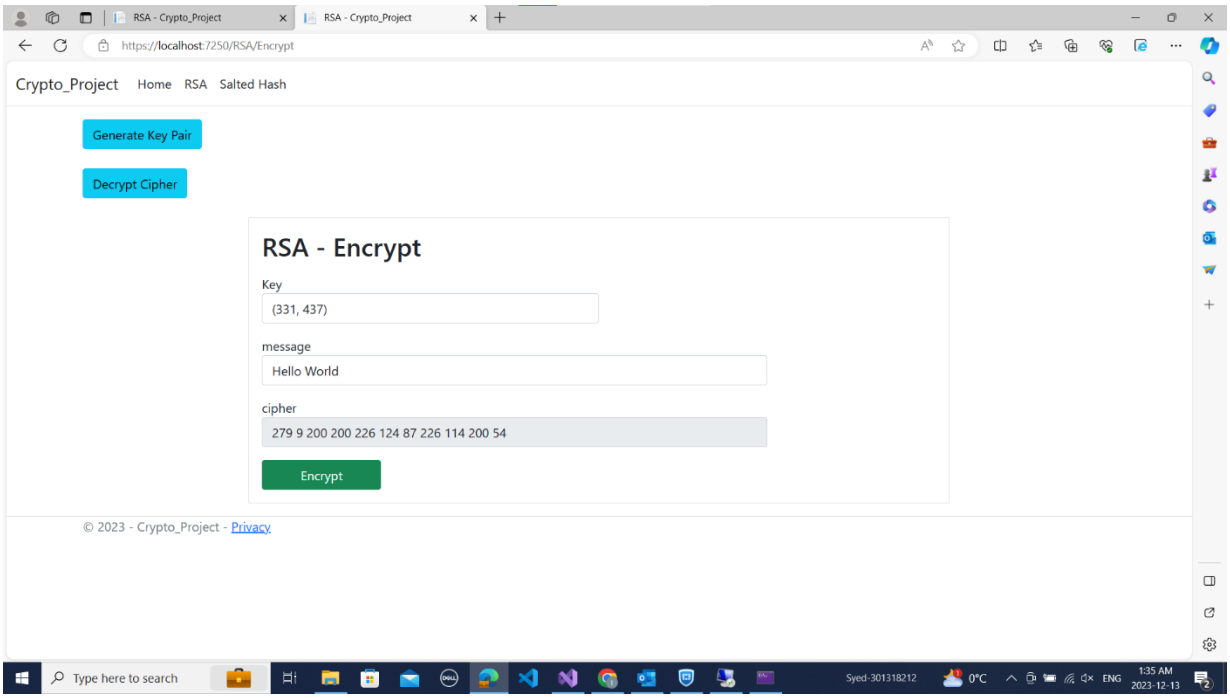

Figure 2 - RSA Encryption

The recipient uses their private key to decrypt the received message. only the owner, who possesses the private key, can decrypt these messages.

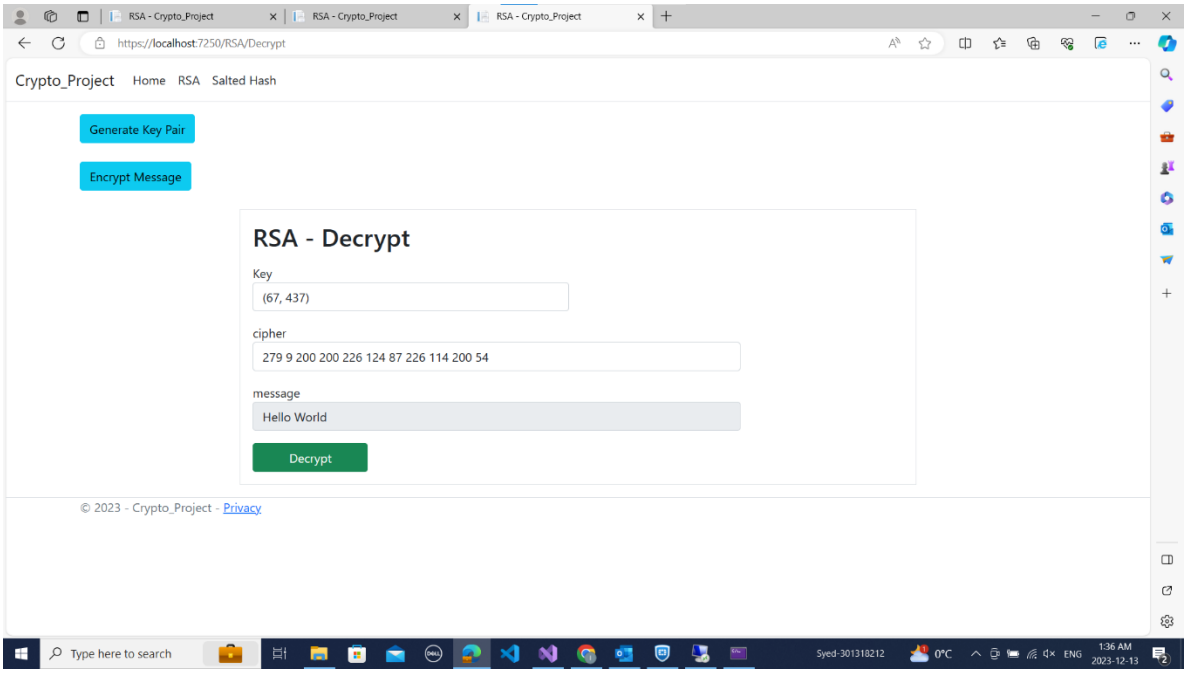

Figure 3 - Decryption RSA

3. Advanced Encryption Standard (AES)

AES is a symmetric encryption algorithm that encrypts and decrypts data. It is widely used to secure sensitive information and communications

The code defines functions for encrypting and decrypting text using the AES (Advanced Encryption Standard) algorithm in CBC (Cipher Block Chaining) mode with a 256-bit key. The 'EncryptAES' function takes a key and plaintext, converts them to byte arrays, encrypts the data using AES, and returns the Base64-encoded ciphertext. The 'DecryptAES' function reverses the process, converting the Base64-encoded ciphertext to bytes, decrypting it using AES, and returning the original plaintext. The code ensures security by using a fixed initialization vector (IV) and adheres to AES best practices for secure symmetric-key encryption in .NET applications.

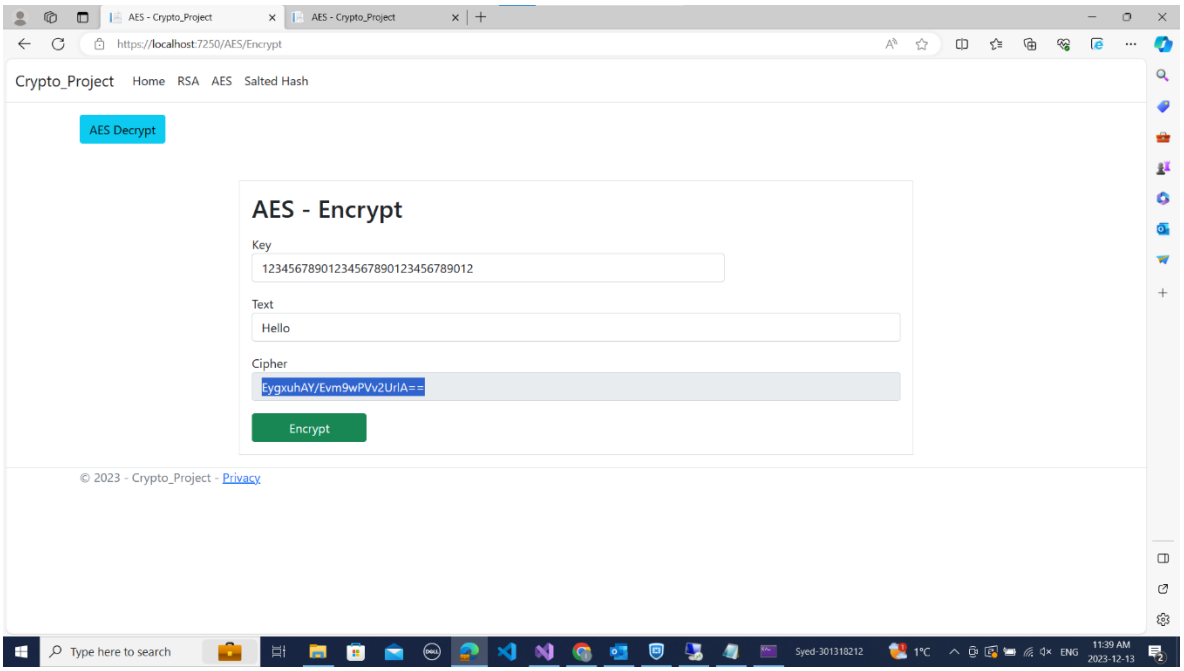

Figure 4 - Encryption AES

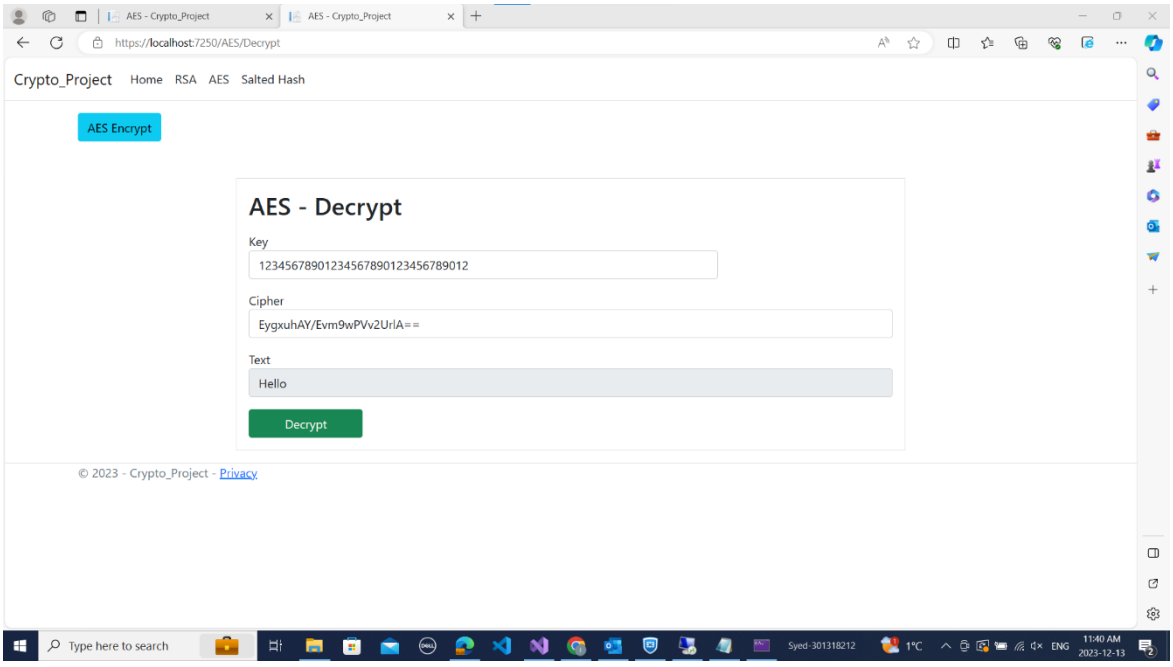

Figure 5 - Decryption AES

4. Salted Hash

A salted hash is a security measure used in password storage to enhance the security of hashed passwords. Hashing is a one-way process that converts input data, such as a password, into a fixed-length string of characters.

The code provides functions for secure password hashing and verification. The 'HashPassword' function takes a user password, generates a random 32-byte salt, and computes the PBKDF2 hash using SHA256 with 350,000 iterations. The result is returned as a hexadecimal string, and the salt is passed as an output parameter. The 'VerifyPassword' function takes a user-entered password, a stored hash, and the corresponding salt. It recomputes the hash using the provided password and salt, then securely compares it to the stored hash using fixed-time comparison to mitigate timing attacks. This ensures robust password security by employing key derivation and salting.

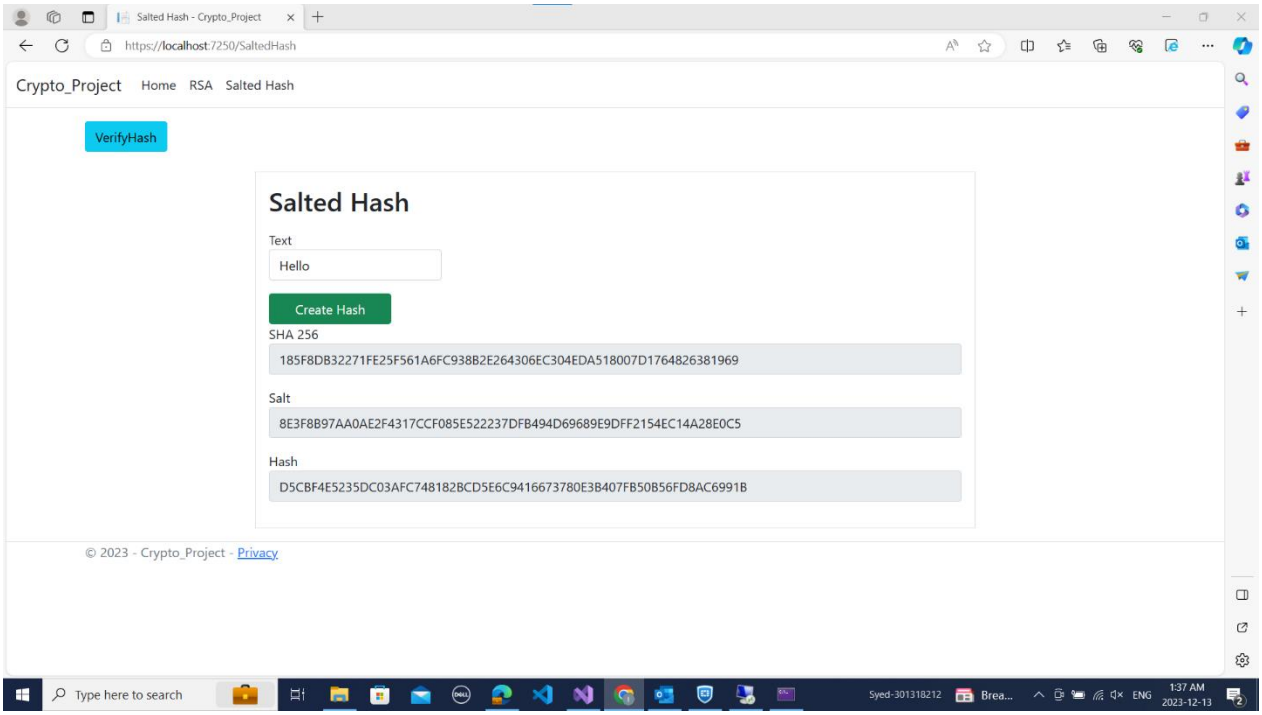

Figure 6 - Create Hash

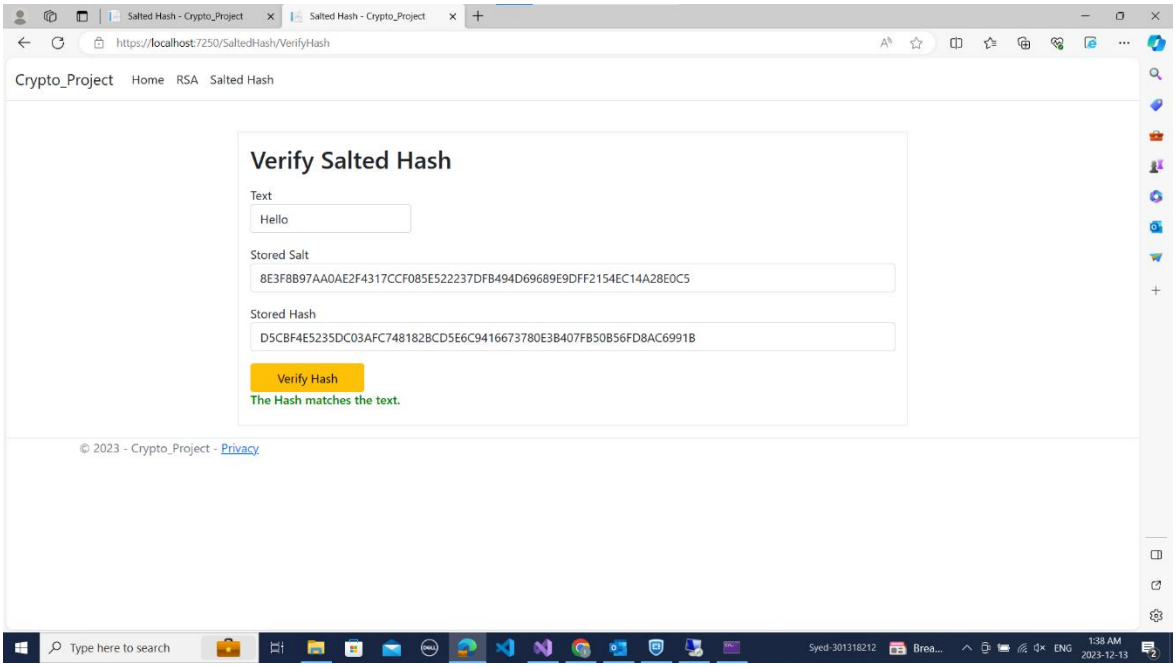

Figure 7 - Salt and Hash matches the Text

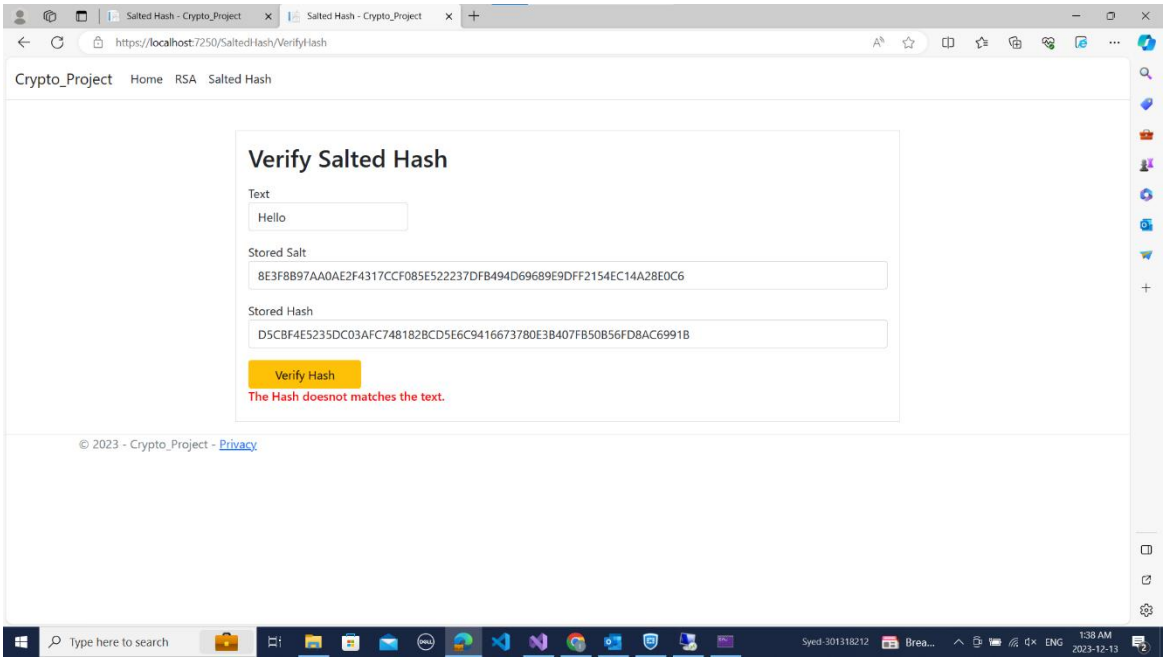

Figure 8 - Salt and Hash does not match the Text

5. Limitations

1. **Random Selection of e and d Parameters:**
    - In the current implementation, the application lacks user input for the selection of the e and d parameters. This could be seen as a limitation in scenarios where users may want to specifically choose these parameters.

- A possible enhancement would be to allow users to input their preferred values for e and d. This can offer more flexibility and control over the encryption and decryption processes.

2. **Handling Errors for Non-Prime Numbers (P or Q):**
    - The application currently lacks error handling for non-prime numbers entered as values for P or Q. This is a potential vulnerability as these values are crucial for the security of the encryption algorithm.
    - To address this limitation, the application should include validation checks to ensure that the values entered for P and Q are indeed prime numbers. If a non-prime number is detected, the application should prompt the user to enter a valid prime number.
3. **Padding for AES Key Length Less Than 32 Characters:**
    - The application does not handle the situation where the key for AES encryption is less than 32 characters. This could lead to security issues as a shorter key might be more susceptible to brute force attacks.
    - To enhance security, the application should include a padding mechanism to ensure that the key length is always 32 characters. This can be achieved by appending or prepending additional characters to the key until it reaches the desired length. Padding schemes like PKCS#7 can be employed for this purpose.

6. Conclusion

In conclusion, this report underscores the significance of cryptographic protocols in safeguarding sensitive information. While the RSA, AES, and Salted Hash implementations exhibit commendable security measures, the outlined limitations necessitate attention for comprehensive robustness. User-centric enhancements in parameter selection, prime number validation, and key length padding have been proposed to fortify these cryptographic practices. By addressing these considerations, the overall security and reliability of the implemented cryptographic algorithms can be elevated, ensuring their efficacy in modern information security landscapes.

7. References

Dotnet-Bot. (n.d.). *System.security.cryptography namespace*. Microsoft Learn. https://learn.microsoft.com/en-us/dotnet/api/system.security.cryptography?view=net-8.0

Hughes, A. (2022, March 1). *Encryption vs. Hashing vs. salting - what's the difference?* Encryption vs. Hashing vs. Salting - What's the Difference? | Ping Identity. https://www.pingidentity.com/en/resources/blog/post/encryption-vs-hashing-vs-salting.html

Wang, S. (2019, August 8). *The difference in five modes in the AES encryption algorithm*. Highgo Software Inc. https://www.highgo.ca/2019/08/08/the-difference-in-five-modes-in-the-aes-encryption-algorithm/

Wickramasinghe, S. (2023, May 15). *RSA algorithm in cryptography: Rivest Shamir Adleman explained*. Splunk. https://www.splunk.com/en_us/blog/learn/rsa-algorithm-cryptography.html