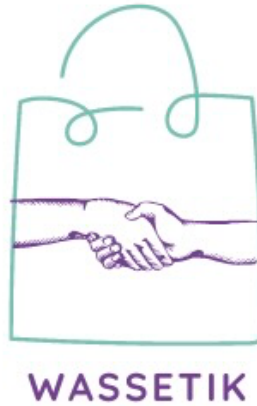


King Saud University
College of Computer and Information Sciences
Department of Software Engineering
SWE 314 – Software Security Engineering
Third Semester 2023



Group#1
Section: 54118
Instructor Name: Dr. Sahar Bayoumi
Submitted on 31 May 2023

#	Student Name	ID
1	Lama Alnasser	442201852
2	Layan Aluwaishiq	442201814
3	Nouv B. Al-Qahtani	442201905
4	Alanoud Alfakhri	442200434
5	Nada Alkubra	442202368

Table of Contents

<i>INTRODUCTION</i>	3
<i>PROJECT SCOPE</i>	3
<i>PRODUCT SCOPE</i>	4
FUNCTIONAL REQUIREMENTS	4
NON-FUNCTIONAL REQUIREMENTS	5
SECURITY REQUIREMENTS	5
<i>OWASP VULNERABILITIES</i>	6
<i>SECURITY CONSIDERATIONS</i>	7
<i>SECURITY TECHNIQUES TO BE IMPLEMENTED</i>	7
<i>IMPLEMENTING A CRYPTOSYSTEM</i>	9
SOURCE CODE	13
<i>REFERENCES</i>	18

INTRODUCTION

Wassetik is a leading e-commerce website that provides customers with a wide variety of products and services at competitive prices and makes purchasing as easy as possible.

PROJECT DESCRIPTION AND IDEA

It is an enterprise for online selling, serving as an intermediary between other retailers and customers. Its web services business includes renting data storage, computing resources, managing payments, and facilitating delivery. The idea behind our website was to create an online marketplace where people could buy and sell goods easily and conveniently.

TARGET USERS

With a global customer base, our website caters to anyone looking to purchase a product or service online, including individuals, families, and businesses seeking to save time, money, and effort by shopping online.

PLATFORM

The system will be programmed using web languages such as HTML, CSS, and JavaScript

PROJECT GOALS AND IMPORTANCE

We aspire to make the customer experience more reliable by protecting the customer and vendor's rights, such as verifying the vendor's information, guaranteeing the right to return and payment.

REFERENCE WEBSITES

- eBay <https://www.ebay.com>
- AliExpress <https://ar.aliexpress.com/>

PROJECT SCOPE

Wassetik is a website that acts as a facilitator between customers and vendors by providing a platform for vendors to display their products to customers. In addition to offering customers a hassle-free method of browsing and buying products from multiple vendors, Wassetik will offer customers a secure payment gateway that supports various payment methods to simplify the purchasing process.

The estimated cost of developing “Wassetik” project using web development languages is around 60,000 SR, and the development period is expected to be approximately 6 months.

PRODUCT SCOPE

FUNCTIONAL REQUIREMENTS

- The Customer:

1. The customer shall be able to create an account using their email address, phone number, and password.
2. The customer shall be able to log in using their credentials (email address and password).
3. The customer shall be able to edit their account information.
4. The customer shall be able to browse all the products.
5. The customer shall be able to view product details (price, description, specifications).
6. The customer shall be able to add products to their shopping cart.
7. The customer shall be able to view their shopping cart.
8. The customer shall be able to modify product quantities.
9. The customer shall be able to remove products from the shopping cart.
10. The customer shall be able to choose from different delivery options, such as standard and express shipping.
11. The customer shall be able to checkout using different payment methods such as Apple Pay, credit card, and PayPal.
12. The customer shall be able to track their order status.
13. The customer shall be able to return products after completing a purchase.
14. The customer shall be able to exchange products after completing a purchase.
15. The customer shall be able to review the products they have purchased.
16. The customer shall be able to rate the products they have purchased.
17. The customer shall be able to communicate with the vendor.
18. The customer shall be able to access their order history.
19. The vendor shall be able to create an account using their email address, phone number, and password.

- The Vendor:

20. The vendor shall be able to log in using their credentials (email address and password).
21. The vendor shall be able to add new products, including (product name, price, specifications, and description)
22. The vendor shall be able to provide an expected delivery date for each product.

- The System:

23. The system should be able to send emails containing ads and offers to customers.
24. The system should check the availability of an item in the shopping cart when it is added to the cart.
25. The system should remove the product from the shopping cart if it is out of stock.
26. The system should inform the customer of the removal of the out-of-stock products from their shopping cart.

NON-FUNCTIONAL REQUIREMENTS

1. The user must be able to login into the system within 1 second.
2. The system downtime should not be more than 1 hour per month.
3. The system should be available to the users 99.9% of the time.
4. The system should be user-friendly (clear, easy to use, and navigate).
5. The system should be available in multiple languages and support different currencies.
6. The system should be able to handle the increased amount of data and usage.
7. The system should be capable of handling a high volume of transactions, 1000 or more simultaneously while maintaining a high level of security.
8. The system should be accurate in order fulfillment, ensuring each customer gets their order.
9. The system shall be able to load all products as the user scrolls down within 1.5 seconds.
10. The system shall be able to integrate with third-party tools such as payment gateways.
11. The system shall be able to handle 1,000,00 concurrent users while maintaining optimal performance.

SECURITY REQUIREMENTS

1. The system must be able to encrypt all personally identifiable information before storing it in the database.
2. The system should remain accessible and operate dependably, even when subjected to denial-of-service attacks.
3. The system payment processing gateway must be PCI DSS (Payment Card Industry Data Security Standard) compliant.
4. The system shall have authentication measures at all the entry points, front panels, or inbound network connections to avoid unauthorized access.
5. The backup system shall store the recovered data in a network system to help in case of failure or intruder action.
6. The system shall ensure system-level accounts have limited privileges to avoid attackers escalating users' accounts to access administrators' features.

OWASP VULNERABILITIES

1. Cryptographic Failures:

The hashed passwords stored in the database could be vulnerable to cryptographic failures.

2. Broken Access Control:

The customer login credentials stored in the database could be at risk of being vulnerable to broken access control.

3. Identification and Authentication Failures:

User login process, which involves using an email address and password, is vulnerable to identification and authentication failures.

4. SQL Injection:

User inputs during the process of reviewing products could be vulnerable to SQL injection attacks.

5. Software and Data Integrity Failures:

Using modules from untrusted sources in software increases the potential for risks related to software and data integrity failures.

6. Security Logging and Monitoring Failures:

Visible logging to the system could be vulnerable to security logging and monitoring failures risk.

7. Server-Side Request Forgery:

If an attacker can manipulate the URL of a third-party service that a web application uses, it may be vulnerable to server-side request forgery.

8. Security Misconfiguration:

Displaying an error message when the user inputs invalid data could be vulnerable to security Misconfiguration.

SECURITY CONSIDERATIONS

1. The *integrity* and *confidentiality* of the hashed passwords stored in the database should be preserved.
2. The *integrity* and *confidentiality* of customer login credentials stored in the database should be preserved.
3. The *authenticity* and the *confidentiality* of user login credentials should be preserved
4. The *integrity* and *confidentiality* of user inputs during the reviewing process should be preserved.
5. The *integrity* of software should be upheld when relying on modules from untrusted sources.
6. The *Integrity* and *Confidentiality* of the system should be protected.
7. a. The user's *authenticity* should be preserved. b. Secure communication *confidentiality* should be preserved.
8. The system's private information *confidentiality* should be preserved.

SECURITY TECHNIQUES TO BE IMPLEMENTED

1. Cryptographic Failures:

To preserve the integrity and the confidentiality of hashed passwords stored in the database, the following security techniques can be implemented:

- a. **Strong Password Hashing Algorithms:** Implement robust cryptographic hash functions like bcrypt, Argon2, or scrypt [1] to protect hashed passwords. These algorithms resist brute-force and dictionary attacks.
- b. **Salted Hashes:** Generate unique random salts for each user and combine them with passwords before hashing. Salting prevents attackers from using precomputed tables and enhances password security.

2. Broken Access Control:

To preserve the integrity and confidentiality of customer login credentials stored in the database and mitigate the risks associated with Broken Access Control, the following security technique can be implemented:

- a. **Principle of Least Privilege (PoLP)** [2]: Follow the PoLP, means users should only be granted the minimum privileges necessary to perform their tasks.

3. Identification and Authentication Failures:

To preserve the authenticity and the confidentiality of user login credentials, including email addresses and passwords, the following security techniques can be implemented:

- a. **Strong Password Policies:** Enforce the usage of strong passwords that meet specific complexity requirements, such as a minimum length and a combination of uppercase and lowercase letters, numbers, and special characters. This helps prevent easy guessing or brute-force attacks on user passwords.
- b. **Multi-Factor Authentication (MFA)** [3]: Implement MFA by requiring users to provide something they know (password) and something they have (one-time password sent to their mobile device) or something they are (biometric authentication like fingerprint or facial recognition).

4. SQL Injection:

To preserve the integrity and confidentiality of user inputs during the reviewing process, the following security technique can be implemented:

a. **Input Validation and Sanitization:** Implement comprehensive input validation and sanitization mechanisms to ensure that user inputs are validated for expected formats and sanitized to eliminate any potentially malicious content.

5. Software and Data Integrity Failures:

To verify the software or data is from the expected source and has not been altered we can use digital signatures or similar mechanisms.

a. **Digital Signing Process** the software or data is signed using a cryptographic algorithm by the entity that created it, typically the software developer or data owner. The signing process involves generating a unique digital signature based on the contents of the software or data.

b. **Signature Verification** to verify the integrity and authenticity of the software or data, the digital signature is checked using the corresponding public key. If the signature is valid, it means the software or data has not been tampered with since it was signed and that it indeed originated from the expected source.

6. Security Logging and Monitoring Failures:

a. By using **Snort** tool to generates detailed logs that capture network traffic and any suspicious or malicious activities it detects. These logs contain valuable information about the source and destination IP addresses, timestamps, protocols used, and specific rules triggered by the detected activity.

b. **SIEM** systems provide a range of additional features that further enhance the detection and response capabilities for logging attacks, SIEM systems have more features to help detect and respond to logging attacks, and detect unusual user behavior. These capabilities enhance the ability to identify and address logging attacks.

7. Server-Side Request Forgery:

a. Input validation by validating all user input and ensuring that it is within the expected range of values before making a request. This can help prevent attackers from manipulating the URL to inject malicious code or execute unauthorized actions.

b. Applying Secure communication by using secure communication protocols (such as HTTPS) to encrypt data in transit and prevent attackers from intercepting or modifying requests.

8. Security Misconfiguration:

Limit error messages by Ensuring that error messages are concise and do not reveal sensitive information about the system or underlying code. Avoid displaying detailed error messages and other technical details that are useful to an attacker and could be used to exploit vulnerabilities in the system.

IMPLEMENTING A CRYPTOSYSTEM

Implementation of the RSA Cryptosystem

The RSA cryptosystem is implemented using HTML, CSS, and JavaScript to provide a user-friendly interface for encryption and decryption. The home page of SecureCrypt features a logo, paragraph, and two call-to-action buttons. The paragraph provides an overview of data security and the importance of RSA encryption. The buttons allow users to access encryption and decryption functionality, as well as algorithm specifications.

The implementation follows standard RSA steps for key generation, encryption, and decryption. Public and private keys are generated using prime numbers and modular arithmetic. The program validates user input and performs encryption and decryption operations within the specified range.

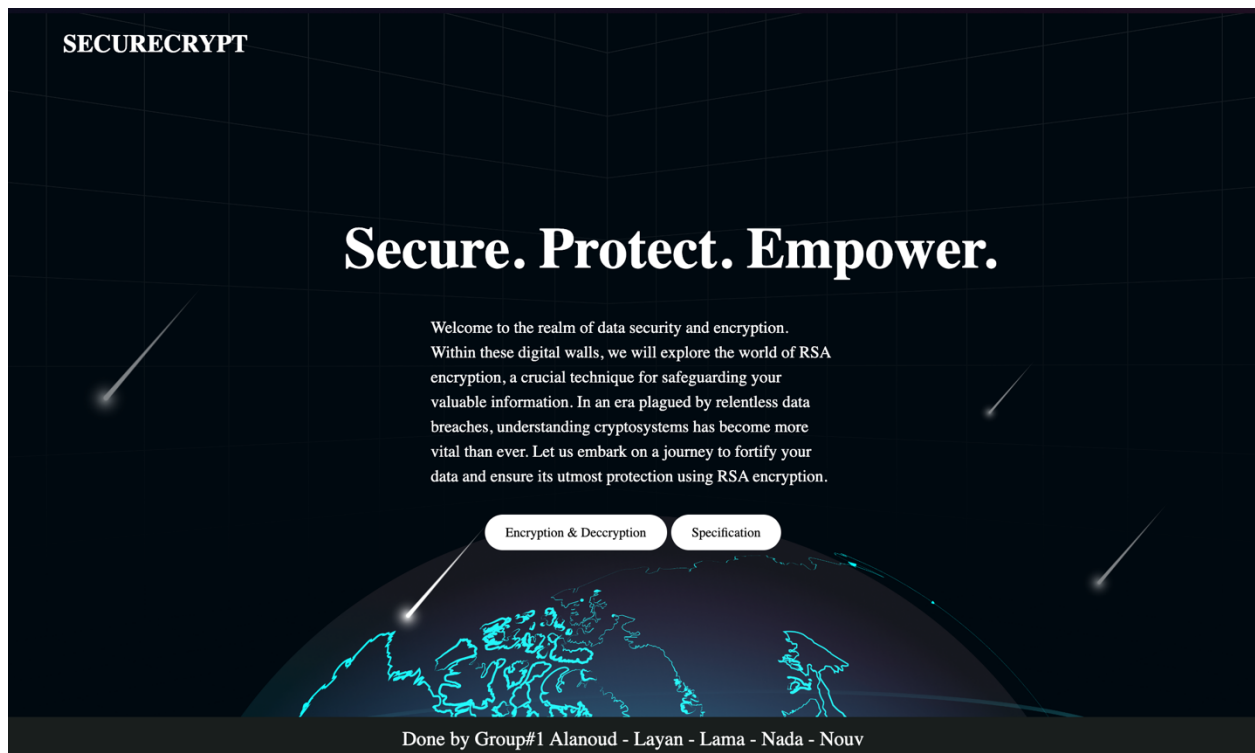
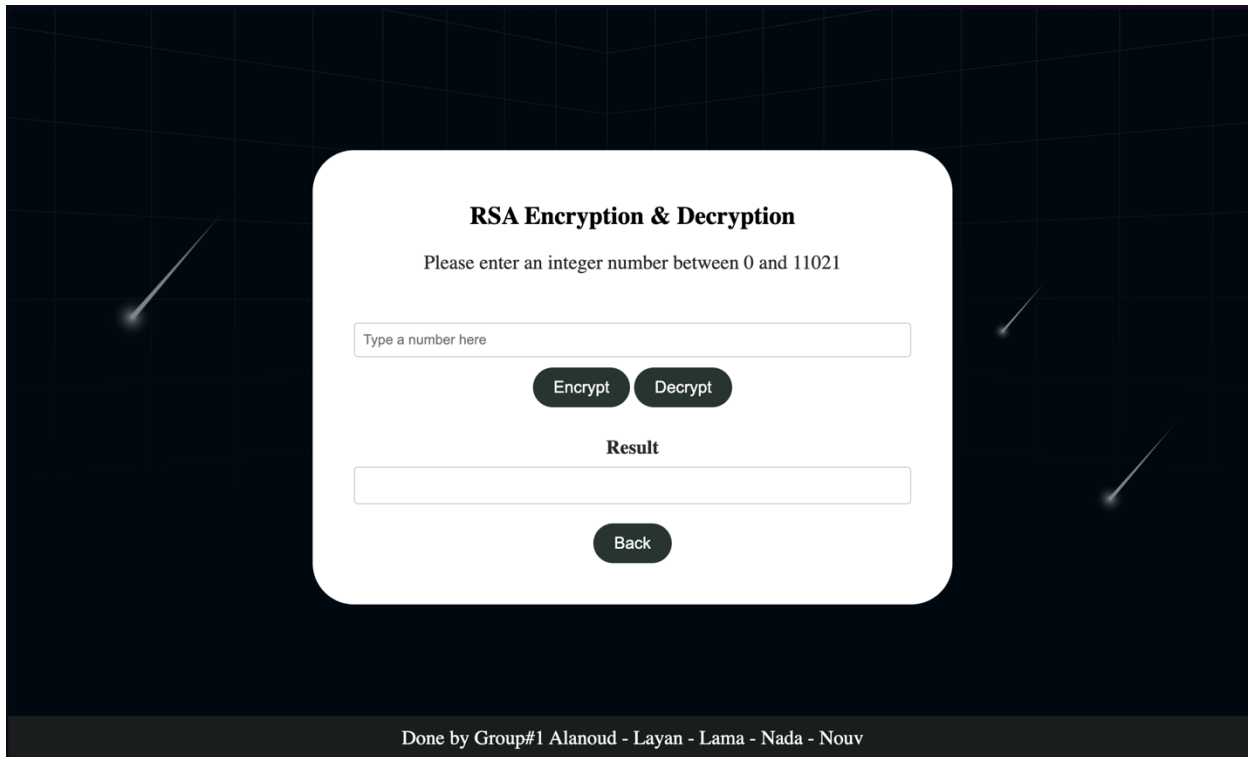


Figure 1 Home Page



RSA Encryption & Decryption

Please enter an integer number between 0 and 11021

Type a number here

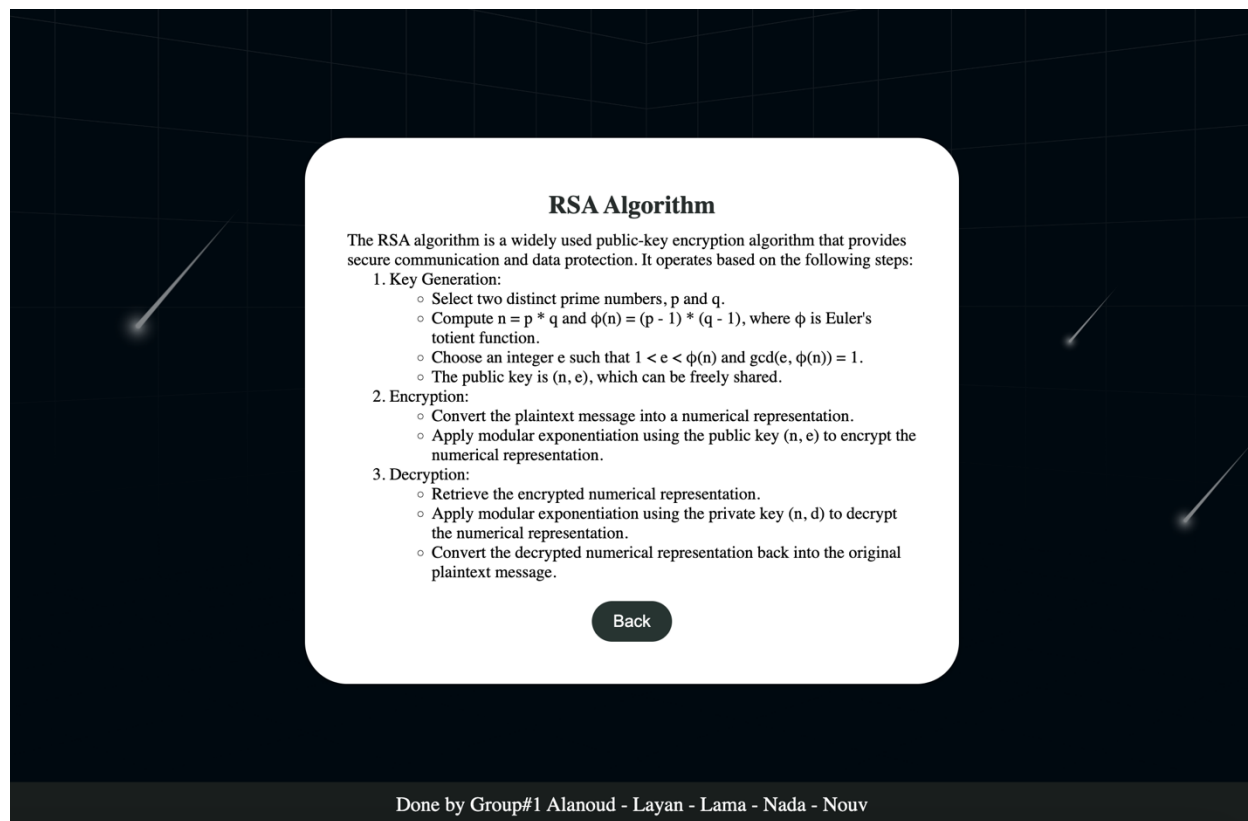
Encrypt Decrypt

Result

Back

Done by Group#1 Alanoud - Layan - Lama - Nada - Nouv

Figure 2 Encryption And Decryption Page



RSA Algorithm

The RSA algorithm is a widely used public-key encryption algorithm that provides secure communication and data protection. It operates based on the following steps:

1. Key Generation:
 - Select two distinct prime numbers, p and q .
 - Compute $n = p * q$ and $\phi(n) = (p - 1) * (q - 1)$, where ϕ is Euler's totient function.
 - Choose an integer e such that $1 < e < \phi(n)$ and $\gcd(e, \phi(n)) = 1$.
 - The public key is (n, e) , which can be freely shared.
2. Encryption:
 - Convert the plaintext message into a numerical representation.
 - Apply modular exponentiation using the public key (n, e) to encrypt the numerical representation.
3. Decryption:
 - Retrieve the encrypted numerical representation.
 - Apply modular exponentiation using the private key (n, d) to decrypt the numerical representation.
 - Convert the decrypted numerical representation back into the original plaintext message.

Back

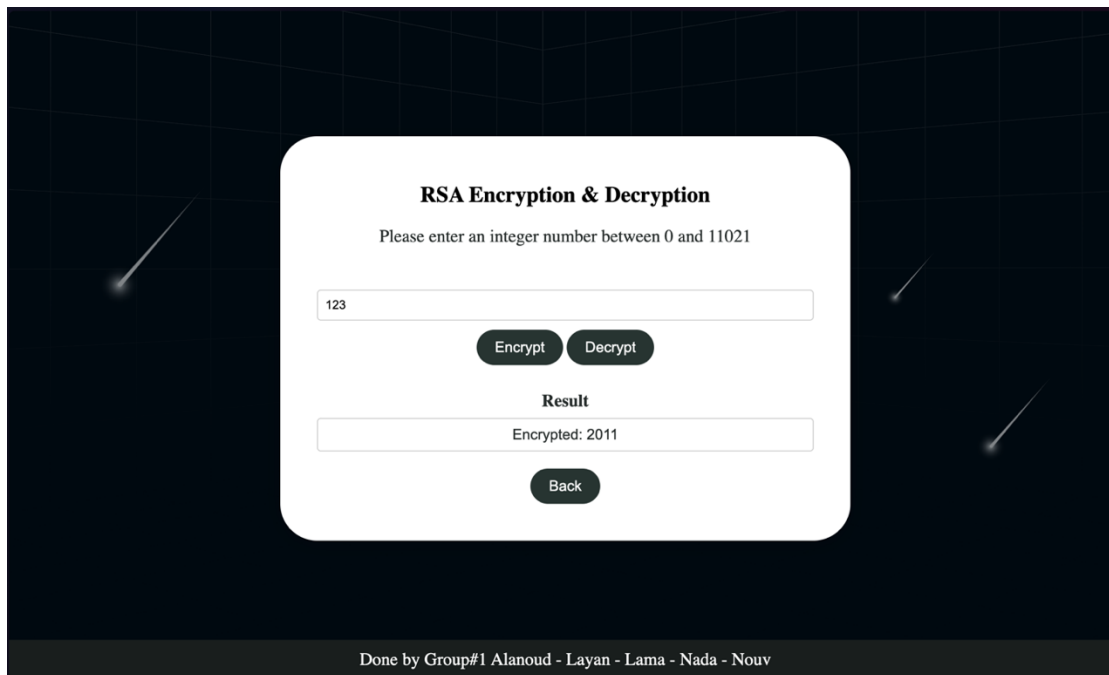
Done by Group#1 Alanoud - Layan - Lama - Nada - Nouv

Figure 3 Algorithm Specifications Page

Tests were conducted using different input values, resulting in successful encryption and decryption. Screenshots demonstrate the program's functionality, showcasing encrypted and decrypted results.

Encryption Test Case:

Test Input: 123 and the output: 2011

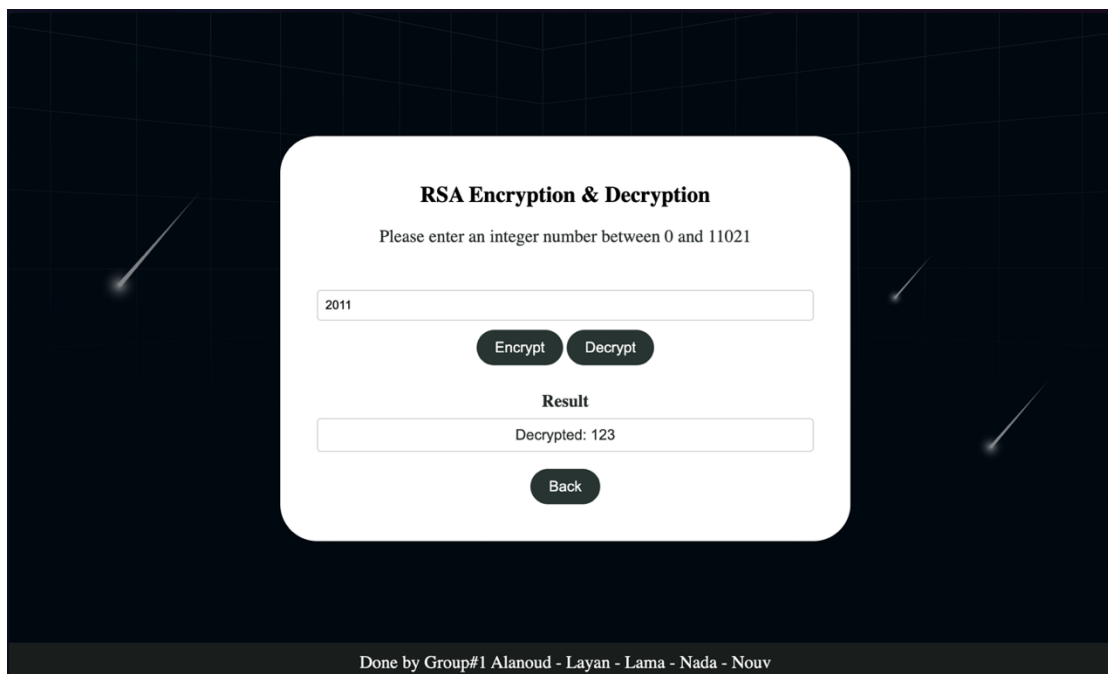


The screenshot shows a web application titled "RSA Encryption & Decryption". Below the title is a prompt: "Please enter an integer number between 0 and 11021". A text input field contains the number "123". Below the input field are two buttons: "Encrypt" and "Decrypt". The "Encrypt" button is highlighted. Below these buttons is a section titled "Result" with a text input field containing "Encrypted: 2011". At the bottom of the result section is a "Back" button. The entire interface is set against a dark background with a subtle grid pattern and some light streaks. At the very bottom, a footer reads "Done by Group#1 Alanoud - Layan - Lama - Nada - Nouv".

Figure 4 Test Case 1

Decryption Test Case:

Test Input: 2011 and output: 123



The screenshot shows the same web application as Figure 4, but with the "Decrypt" button highlighted. The text input field now contains the number "2011". The "Result" section's text input field now displays "Decrypted: 123". The "Back" button remains at the bottom of the result section. The background and footer are identical to the previous screenshot.

Figure 5 Test Case 2

During the testing phase, invalid test cases, including non-numeric values and numbers out of range, should be considered to ensure the robustness of the system. Here are some examples of such test cases:

Non-numeric values:

Test Input: "swe" **Expected behavior:** The system should detect that the input is not a valid number and display an alert message saying, "Please enter a valid integer between 0 and 11021."

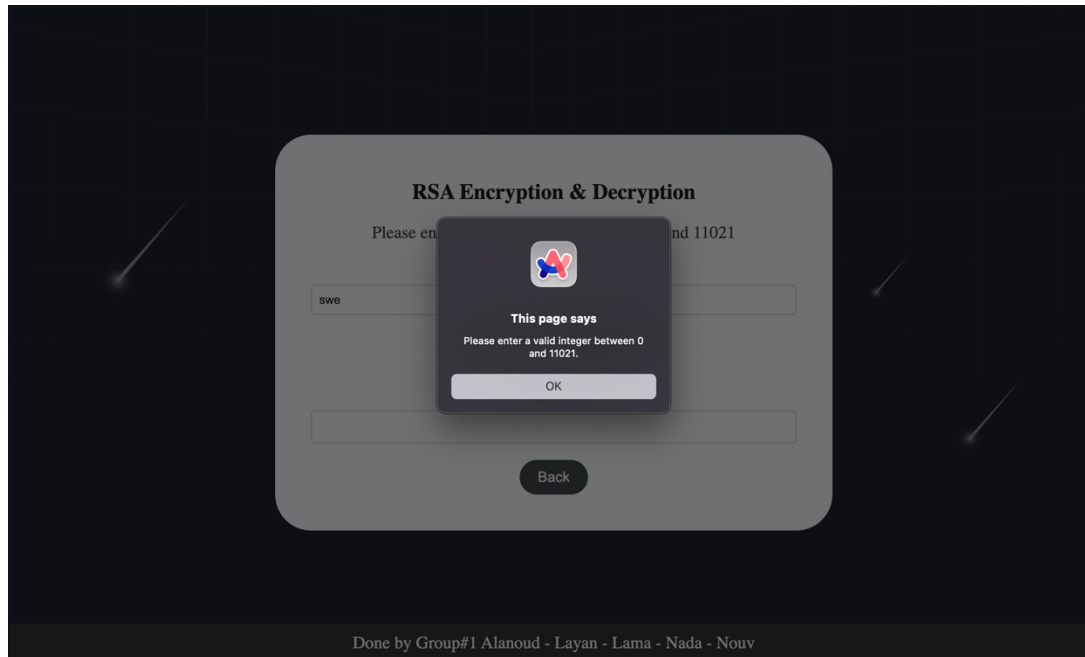


Figure 6 Test Case 3

Number out of range:

Test Input: 929999 **Expected behavior:** The system should detect that the input is not a valid number and display an alert message saying, "Please enter a valid integer between 0 and 11021."

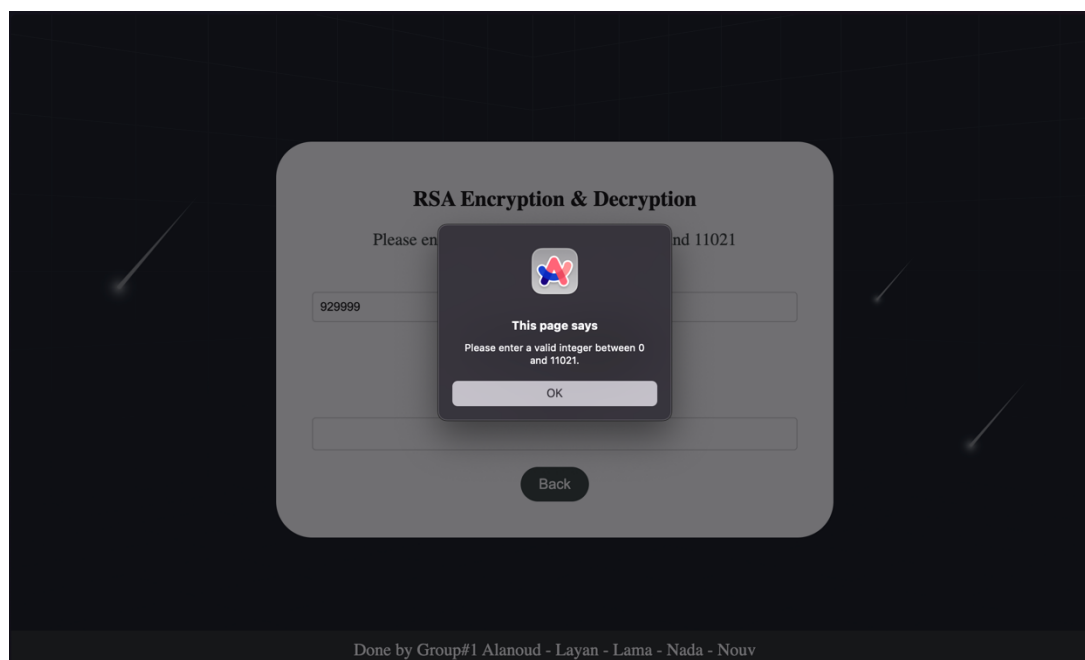


Figure 7 Test Case 3

SOURCE CODE

1. encryptNumber(): This function is triggered when the "Encrypt" button is clicked. It retrieves the input number from the numberInput element, validates it to ensure it is a valid integer within the specified range, and performs RSA encryption on the number. The encrypted result is then displayed in the resultOutput element.

```
10 <script>
11 // RSA Encryption & Decryption
12 function encryptNumber() {
13   var numberInput = document.getElementById("numberInput").value;
14   var resultOutput = document.getElementById("resultOutput");
15
16   if (!/^[+-]?\d+$/ .test(numberInput) || numberInput < 0 || numberInput > 11021) {
17     alert("Please enter a valid integer between 0 and 11021.");
18     return;
19   }
20
21   var number = parseInt(numberInput);
22
23   var p = 103; // Prime number p
24   var q = 107; // Prime number q
25
26   // Calculate n
27   var n = p * q;
28
29   // Calculate phi
30   var phi = (p - 1) * (q - 1);
31
32   // Calculate public key exponent e
33   var e = calculatePublicKeyExponent(phi);
34
35   // Calculate private key exponent d
36   var d = calculatePrivateKeyExponent(e, phi);
37
38   var encryptedNumber = modPow(number, e, n);
39   resultOutput.value = "Encrypted: " + encryptedNumber;
40 }
```

Figure 7 Code Implementation

2. decryptNumber(): This function is triggered when the "Decrypt" button is clicked. It retrieves the input number from the numberInput element, validates it to ensure it is a valid integer within the specified range, and performs RSA decryption on the number. The decrypted result is then displayed in the resultOutput element.

```
41
42 function decryptNumber() {
43     var numberInput = document.getElementById("numberInput").value;
44     var resultOutput = document.getElementById("resultOutput");
45
46     if (!/^[--+]?[0-9]+$/.test(numberInput) || numberInput < 0 || numberInput > 11021) {
47         alert("Please enter a valid integer between 0 and 11021.");
48         return;
49     }
50
51     var number = parseInt(numberInput);
52
53     var p = 103; // Prime number p
54     var q = 107; // Prime number q
55
56     // Calculate n
57     var n = p * q;
58
59     // Calculate phi
60     var phi = (p - 1) * (q - 1);
61
62     // Calculate public key exponent e
63     var e = calculatePublicKeyExponent(phi);
64
65     // Calculate private key exponent d
66     var d = calculatePrivateKeyExponent(e, phi);
67
68     var decryptedNumber = modPow(number, d, n);
69     resultOutput.value = "Decrypted: " + decryptedNumber;
70 }
71
```

Figure 8 Code Implementation

3. calculatePublicKeyExponent(phi): This function calculates the public key exponent (e) used in the RSA encryption. It starts with a default value of 17 and checks if it satisfies the conditions of being coprime with phi. If the default value doesn't satisfy the conditions, it finds the next coprime value of e and returns it.

4. calculatePrivateKeyExponent(e, phi): This function calculates the private key exponent (d) used in the RSA decryption. It uses the Extended Euclidean Algorithm to find the modular multiplicative inverse of e modulo phi. It ensures that d is positive and returns it.

```
72
73  function calculatePublicKeyExponent(phi) {
74      // Choose a public key exponent e such that 1 < e < phi and e is coprime with phi
75      var e = 17; // Default value for it can be changed
76
77      // Check if the default value satisfies the conditions
78      if (isCoprime(e, phi)) {
79          return e;
80      }
81
82      // Find the next coprime value of e
83      while (!isCoprime(e, phi)) {
84          e++;
85      }
86
87      return e;
88  }
89
90  function calculatePrivateKeyExponent(e, phi) {
91      // Calculate the modular multiplicative inverse of e modulo phi
92      // Here, we'll use the Extended Euclidean Algorithm to find the modular inverse
93      var d = extendedEuclidean(e, phi);
94
95      // Make sure d is positive
96      if (d < 0) {
97          d = (d % phi + phi) % phi;
98      }
99
100     return d;
101 }
102
```

Figure 9 Code Implementation

5. isCoprime(a, b): This function checks if two numbers, a and b, are coprime, (their greatest common divisor) is 1. It uses the Euclidean Algorithm to calculate the GCD iteratively. If the GCD is 1, it returns true; otherwise, it returns false.

6. extendedEuclidean(a, b): This function calculates the modular inverse of a modulo b using the Extended Euclidean Algorithm. It returns the modular inverse.

```
103 function isCoprime(a, b) {
104     // Check if a and b are coprime (their greatest common divisor is 1)
105     while (b !== 0) {
106         var temp = b;
107         b = a % b;
108         a = temp;
109     }
110     return a === 1;
111 }
112
113 function extendedEuclidean(a, b) {
114     if (b === 0) {
115         return a;
116     }
117     var x1 = 1;
118     var y1 = 0;
119     var x2 = 0;
120     var y2 = 1;
121
122     while (b !== 0) {
123         var q = Math.floor(a / b);
124         var r = a % b;
125
126         var x = x1 - q * x2;
127         var y = y1 - q * y2;
128
129         a = b;
130         b = r;
131
132         x1 = x2;
133         y1 = y2;
134         x2 = x;
135         y2 = y;
136     }
137     return x1;
138 }
```

Figure 10 Code Implementation

7. modPow(base, exponent, modulus): This function performs modular exponentiation. It calculates base raised to the power of exponent, modulo modulus. It uses the binary exponentiation algorithm to efficiently calculate the result. The implementation of this function was taken from the following source: [<https://umaranis.com/2018/07/12/calculate-modular-exponentiation-powermod-in-javascript-ap-n/>].

```
140     function modPow(base, exponent, modulus) {  
141         if (modulus === 1) return 0;  
142         var result = 1;  
143         base = base % modulus;  
144         while (exponent > 0) {  
145             if (exponent % 2 === 1)  
146                 result = (result * base) % modulus;  
147             exponent = Math.floor(exponent / 2);  
148             base = (base * base) % modulus;  
149         }  
150         return result;  
151     }
```

Figure 11 Code Implementation

REFERENCES

[1] Stytech, "Argon2 vs bcrypt vs scrypt," Stytech Blog, March 1, 2023. [Online]. Available: <https://stytech.com/blog/argon2-vs-bcrypt-vs-scrypt/>

[2] M. Marian, "What Is Broken Access Control?," Heimdal Security Blog, Mar. 1, 2023. [Online]. Available: <https://heimdalsecurity.com/blog/what-is-broken-access-control/>. [Accessed: May 27, 2023].

[3] J. Tullman-Botzer, "Identification and Authentication Failures and How to Prevent Them," Cyolo Blog, May 11, 2022. [Online]. Available: <https://cyolo.io/blog/identification-and-authentication-failures-and-how-to-prevent-them>. [Accessed: 26 May 2023].