University of San Carlos
School of Arts and Sciences
Department of Computer, Information Sciences, and Mathematics
Talamban Campus, Cebu City, Philippines

IT 3104N – Information Assurance and Security

PROJECT: CRYPTOGRAPHY

Submitted by:
Van AJ Vanguardia

Submitted to:
Godwin Monserate

05 December 2022

Overview

The project is a combination between a modified RSA (Rivest-Shamir-Adleman) and the Atbash cipher. RSA proposed a method for implementing a public-key cryptosystem whose security rests in part on the difficulty of factoring large numbers. RSA was modified to use four keys of prime numbers (p, q, r, s) instead of two (p, q), which improves security due to its added complexity.

The Atbash Cipher is a simple substitution cipher that is sometimes called mirror code and it was modified by taking the extra step of reversing the result string after the reversal or mirroring of an alphabet. This project intents to provide a more complex cipher algorithm owing to large factorials and mappings in order to attain more secured encryption and decryption process. Furthermore, the methods and modifications applied in the project are broken down into six sections: key generation, encryption process, decryption process, reverse process, modified RSA and Athbash cipher visualization.

Key Generation

1. Choose four prime numbers where P, Q, R, S are distinct
2. Calculate N (Product of the 4 prime numbers)
    - Let $N = p * q * r * s$
3. Calculate T (Totient of Product of Primes)
    - Let $T = (p -1) * (q - 1) * (r - 1) * (s - 1)$
4. Select e (encryption) such that:
    - $1 < e < T$
    - Coprime of N and T or gcd(T, e) = 1;
5. Select d (decryption) such that:
    - $d = de(\text{mod } T) = 1$
6. Generate public key
    - Let KU = {e, N}
7. Generate private Key
    - Let KR = {d, N}

Encryption Process

1. Modified Rivest-Shamir-Adleman – Encryption
   Formula: C = M ^ e mod N
   Where:
   C = Ciphertext
   M = Message
   e = Encryption
   N = Product of the 4 prime numbers

2. Modified Atbash
   (1) First Phase
   Formula: N – L

Where:

N = Max length of the alphabet (25, starting with 0)

L = Index of the alphabet (0 – 25)

Example:

25 – 0 (A) = 25 (Z)

25 – 25 (Z) = 0 (A)

(2) Second Phase

Formula: REV(S)

S = String of atbash characters

REV = String reverse function

Example:

Khoor -> rookH

Decryption Process

1. Modified Atbash

(1) First Phase: Normal Atbash

Formula: N – L

Where:

N = Max length of the alphabet (25, starting with 0)

L = Index of the alphabet (0 – 25)

Example:

25 – 0 (A) = 25 (Z)

25 – 25 (Z) = 0 (A)

(2) Second Phase: Reversal of Atbash

Formula: REV(S)

S = String of atbash characters

REV = String reverse function

Example:

elloH -> Hello

2. Modified Rivest-Shamir-Adleman Cryptography – Decryption

Formula: M = C ^ d mod n

Where:

M = Message

C = Ciphertext

d = Decryption

N = Product of the 4 prime numbers

Reverse Process

Swap the positions of the string element using loop and iterate until i is equals to j.

Formula:

temp = text[i]

text[i] = text[j]

text[j] = temp

Where:

i = 0; starting index of a string

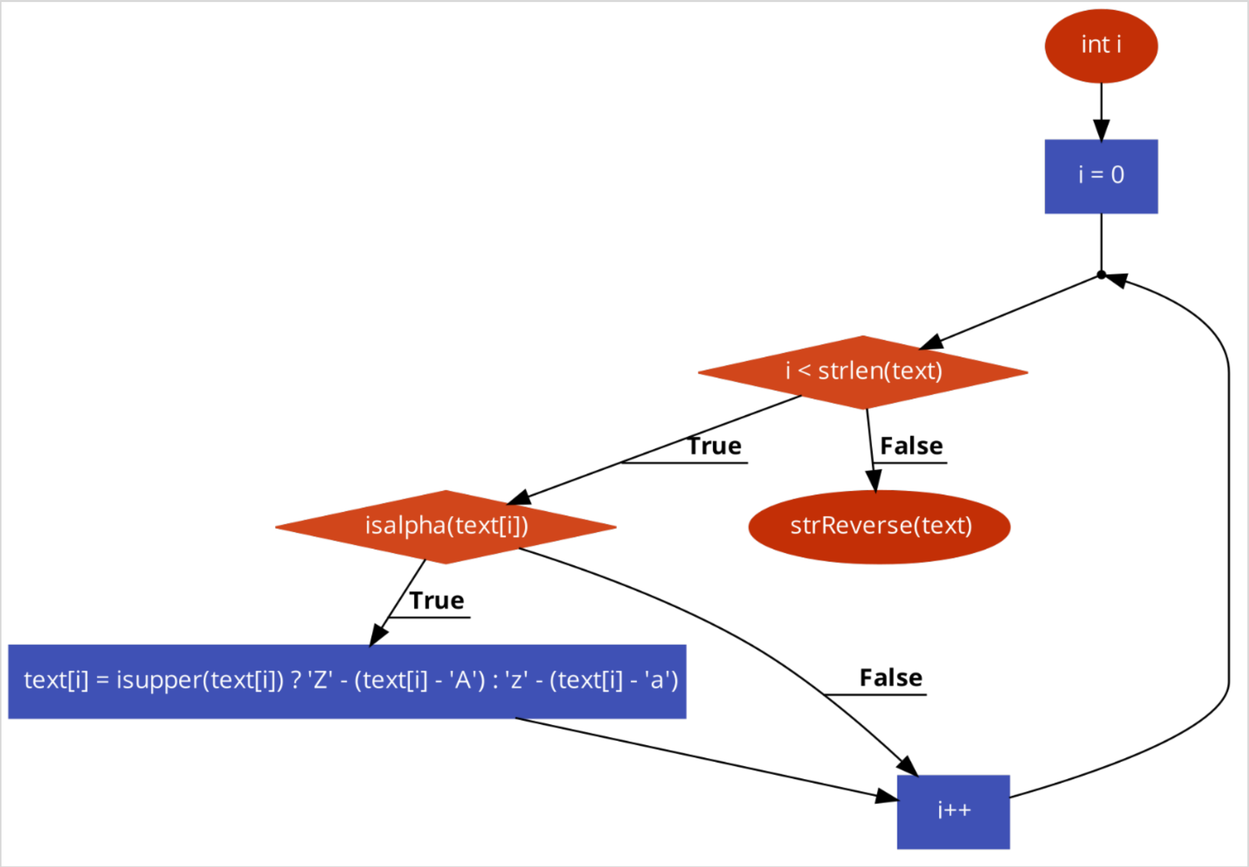
j = len(string); the length of a string

temp = temporary storage of a character

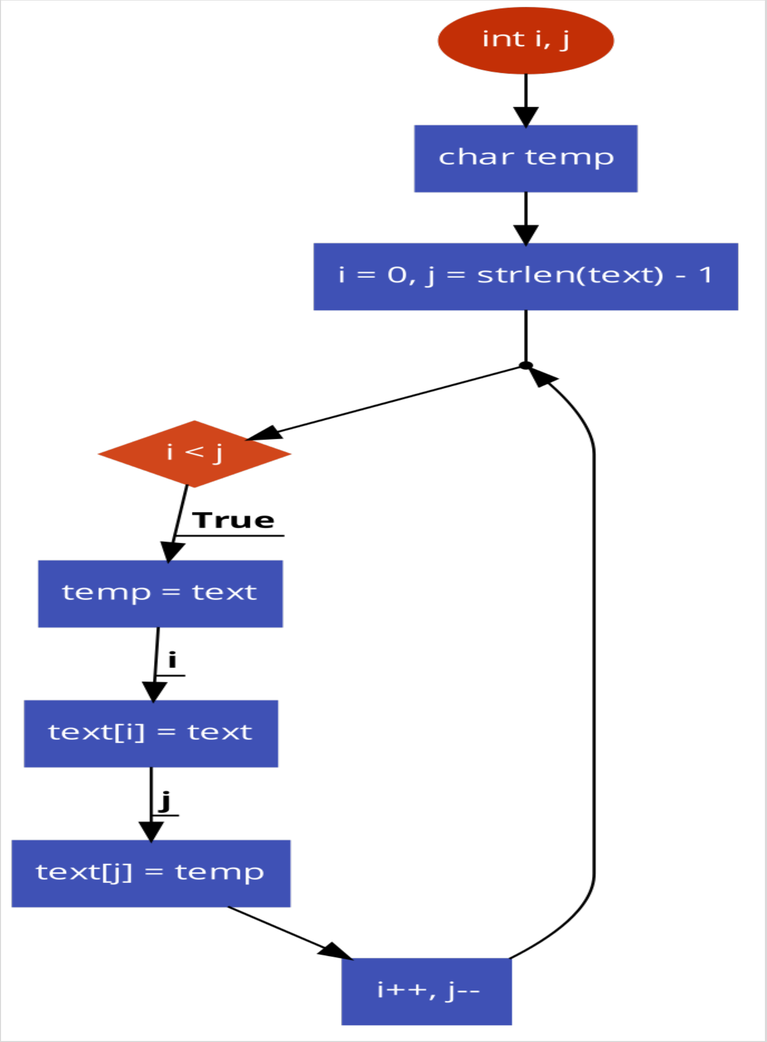
text = array of characters

Modified Atbash – Visualization

The Atbash Cipher is a simple substitution cipher, also known as mirror code, that was enhanced by reversing the result string after an alphabet's reversal or mirroring. The flow and algorithm of the modified Atbash is shown below.



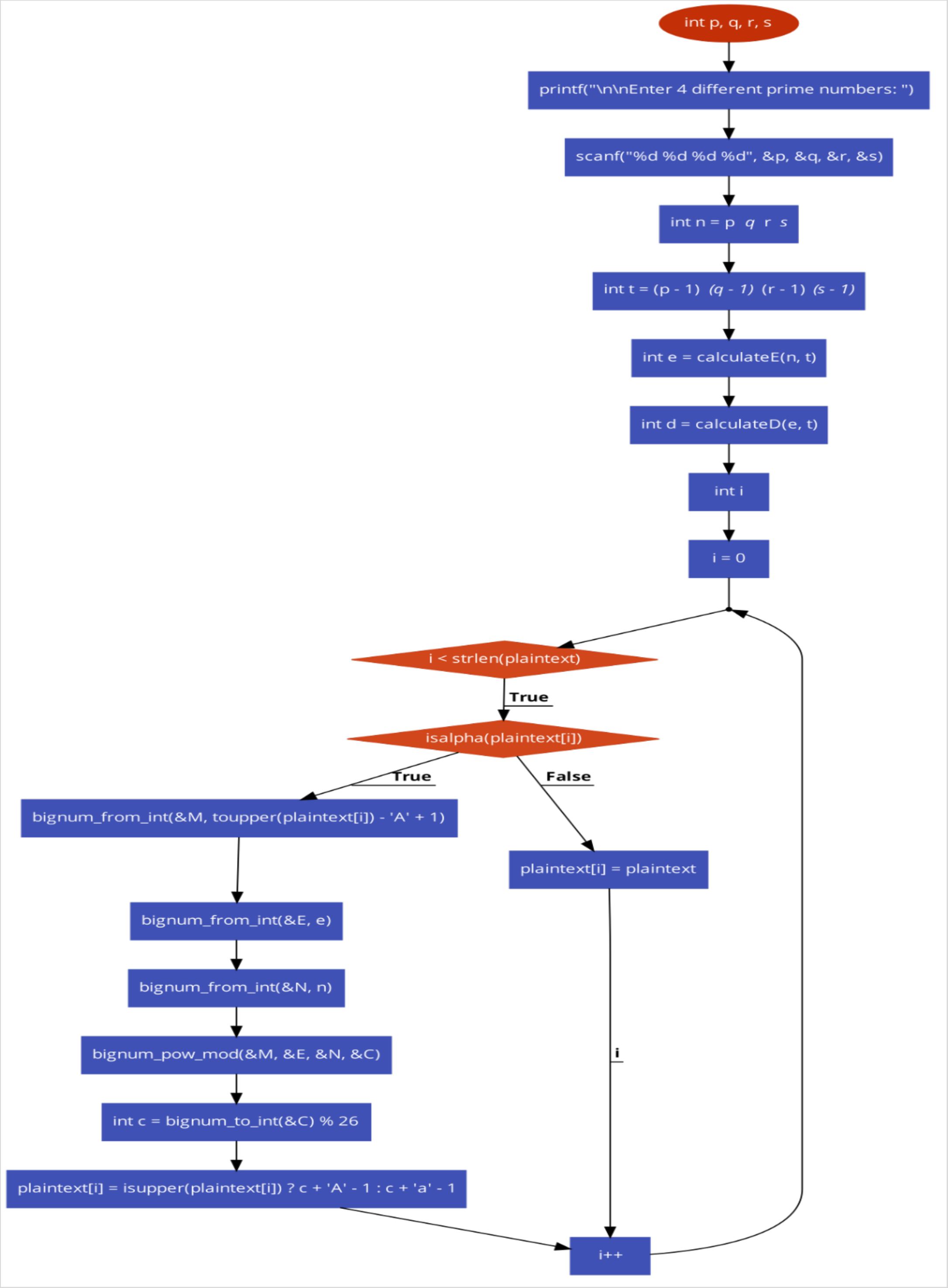
First Phase: Atbash Cipher - Flowchart



Second Phase: String Reverse – Flowchart

Modified RSA – Visualization

Because of the extra complexity, RSA was changed to employ four keys of prime numbers (p, q, r, s) instead of two (p, q). The modified RSA flow and algorithm are shown below.


int p, q, r, s
printf("\n\nEnter 4 different prime numbers: ")
scanf("%d %d %d %d", &p, &q, &r, &s)
int n = p q r s
int t = (p - 1) (q - 1) (r - 1) (s - 1)
int e = calculateE(n, t)
int d = calculateD(e, t)
int i
i = 0
i < strlen(plaintext)
True
isalpha(plaintext[i])
True
False
bignum_from_int(&M, toupper(plaintext[i]) - 'A' + 1)
plaintext[i] = plaintext
bignum_from_int(&E, e)
bignum_from_int(&N, n)
bignum_pow_mod(&M, &E, &N, &C)
int c = bignum_to_int(&C) % 26
plaintext[i] = isupper(plaintext[i]) ? c + 'A' - 1 : c + 'a' - 1
i
i++

Modified RSA – Flowchart