Introduction to Cryptography - Final Documentation

Documentation for RSA homework

Huu Thuan Nguyen

2024-20-01

Contents

1 General information

MSSV	Họ và Tên	Môn học	Giảng viên
21120566	Nguyễn Hữu Thuận	Nhập môn Mã hoá - Mật mã	Đặng Trần Minh Hậu

2 Greatest fulfillments

Điều tâm đắc nhất phải kể đến là được học về môn học, được tạo điều kiện và được trau dồi thêm đam mê của bản thân về lĩnh vực này.

Trong quá trình thực hiện đồ án này, em đã triển khai một thư viện Big Unsigned Integer (số Nguyên Dương Lớn) trong C++ để phục vụ quá trình nhập xuất các Hexadecimal (số Thập lục phân) và các phép tính trên đó.

Thư viện này một phần được xây dựng và tối ưu hoá lại từ một số thư viện khác trên các ngôn ngữ khác nhau, phần còn lại được em triển khai trong thời gian làm đồ án.

3 Difficulties

3.1 Unit-level bugs in Big Unsigned Integer

Trong quá trình thực hiện đồ án này, em đã gặp phải rất nhiều khó khăn, đặc biệt là về vấn đề xử lí các hexadecimal và các phép tính cơ bản trên đó. Vì không được test kĩ càng trước khi triển khai các phép tính nâng cao hơn nên đã mất rất nhiều thời gian để debug các lỗi mà không biết vấn đề nằm ở đâu, những khó khăn này kéo dài xuyên suốt quá trình thực hiện đồ án.

3.2 60-second time limit handling

Một khó khăn khác là yêu cầu tự ngắt chương trình khi thời gian thực hiện vượt quá 60s. Em đã thử nhiều cách nhưng không thể triển khai nó được, dẫn đến việc phải bỏ qua yêu cầu này.

3.3 Vague requirements

Khó khăn còn lại là về vấn đề không đến lớp, em học chủ yếu từ Google và kiến thức cũ, dẫn đến việc cách triển khai của em có thể khác với cách triển khai và yêu cầu của giảng viên. Trong suốt các đồ án em chỉ bám vào yêu cầu của đề bài và làm đúng theo đó, dẫn đến nhiều hệ quả mà chính em cũng đã nhận thức được từ trước nhưng không thể tránh khỏi.

4 Improvement ideas

4.1 Time optimization for Big Unsigned Integer

Thư viện Big Unsigned Integer tuy hoạt động tốt nhưng lại không đạt hiểu quả cao về tốc độ và hiệu suất. Nó đã vượt quá thời gian 60s đối với các test mà số nguyên có độ dài 1024 bit.

Vì vậy, em đề xuất một số ý tưởng cải tiến như sau (xếp theo mức độ ảnh hưởng đến hiệu suất):

- Tạo riêng các method tối ưu cho việc tính toán với số 2 (như phép nhân, chia hoặc lũy thừa) thay vì dùng chung với các method khác. Điều này có thể làm giảm một phần thời gian thực hiện các thuật toán.
- Giảm thiểu các phép tính hoặc phép gán thừa trong các thuật toán, hạn chế sự lặp lại trong code để tối ưu thời gian thực hiện.

- Thử và thay đổi một số thuật toán khác để xem xét hiệu quả về thời gian của chúng.
- Sử dụng hexadecimal cho việc lưu trữ và tính toán thay vì dùng thuật toán thông thường để chuyển đổi nó sang Decimal (hệ Thập phân) rồi chuyển ngược lại.

5 Method/Algorithm implementations

Vì 95% thuật toán và thời gian thực hiện của đồ án này nằm ở thư viện Big Unsigned Integer, trong khi đó các phần khác khá đơn giản và ngắn. Do đó báo cáo này sẽ tập trung hầu hết vào phần thư viện Big Unsigned Integer mà em đã triển khai.

5.1 Big Unsigned Integer library

Chia một số nguyên dương thành các blocks. Mỗi block lưu trữ 6 chữ số thập phân. Do đó, giá trị của một block nằm trong khoảng $[10^6; 10^7 - 1]$ (trừ block cuối cùng).

```
vector<int> data
```

Trên máy 64-bit, kích thước của một `int` là 32(bits) = 4(bytes). Do đó giới hạn kích thước của vector trên là $2^{64}/4 - 1 = 2^{62} - 1$ (phần tử).

Từ đây, ta có thể suy ra được (trong trường hợp lý tưởng):

- Kích thước lưu trữ tối đa $\in [ln(10^6) \times (2^{62} - 1);\ ln(10^7 - 1) \times (2^{62} - 1)] \approx 2^{59}$ (bytes)
- Giá trị lớn nhất $\in [2^{62} \times 10^6;\ 2^{62} \times (10^7 - 1)] \approx 2^{82}$

Lí do tại sao lại chia như vậy là vì thuật toán **Fast Fourier Transform** bên dưới.

5.1.1 Fast Fourier Transform: $A \times B$

Phép nhân 2 số nguyên là một trong những bài toán lớn trong lĩnh vực Mật Mã Học. Nó ảnh hưởng trực tiếp đến hiệu suất của các thuật toán mã hoá, giải mã và các thuật toán khác.

Nhân 2 số nguyên với n chữ số có thể được thực hiện với độ phức tạp $O(n \log(n) \log(\log(n)))$ bằng cách sử dụng Fast Fourier Transform thay vì $O(n^2)$ bằng cách thông thường. Kĩ thuật này sử dụng phương pháp **Polynomial Multiplication** (Nhân Đa thức) và **Recursive Divide and Conquer** (Chia Để Trị).

Một trong những triển khai hiệu quả của Fast Fourier Transform trong việc tìm tích của 2 số nguyên lớn là thuật toán Schönhage–Strassen.

5.1.1.1 Polynomial Multiplication

Ta sẽ coi một số nguyên dương N là một đa thức $P(x)$ thoả mãn $P(x) = \sum_{i=0}^{n-1} a_i x^i$ với a_i là các chữ số trong N. Ta gọi x là Base (Cơ số) của N, a là **Coefficient Vector** (Vector Hệ số) của N.

Ví dụ, số nguyên 123456789 là một đa thức $P(x) = 9x^0 + 8x^1 + 7x^2 + 6x^3 + 5x^4 + 4x^5 + 3x^6 + 2x^7 + 1x^8$ với base $B = 10$ và coefficient vector $a = [9, 8, 7, 6, 5, 4, 3, 2, 1]$.

Ngoài ra:

- Với base $B = 100$, ta sẽ có $a = [89, 67, 45, 23, 1]$
- Với base $B = 1000$, ta sẽ có $a = [789, 456, 123]$
- Với base $B = 16$, ta sẽ có $a = [5, 1, 13, 12, 11, 5, 7]$ (tương đương 0x51d0b57, là biểu diễn hexadecimal của 123456789 dưới dạng little-endian)

Đó là lí do tại sao ta lại chia một số nguyên thành các blocks như đã đề cập.

5.1.1.2 Recursive Divide and Conquer

Chúng ta sẽ chia đa thức $P(x)$ thành 2 đa thức con $P_{even}(x)$ và $P_{odd}(x)$.

Trong đó:

- $P_{even}(x)$ là đa thức con của $P(x)$ với các hệ số ở vị trí chẵn.
- $P_{odd}(x)$ là đa thức con của $P(x)$ với các hệ số ở vị trí lẻ.

Ví dụ đa thức $P(x) = a_0 + a_1x + a_2x^2 + \cdots + a_{n-1}x^{n-1}$ sẽ được chia thành:

$$P_{even}(x) = a_0 + a_2x + a_4x^2 + \cdots + a_{n-2}x^{n/2-1}$$
$$P_{odd}(x) = a_1 + a_3x + a_5x^2 + \cdots + a_{n-1}x^{n/2-1}$$

Khi đó, ta có thể viết lại đa thức $P(x)$ thành:

$$P(x) = P_{even}(x^2) + xP_{odd}(x^2)$$

Cho số phức w là **Primitive n-th Root of Unity** (Căn Đơn vị Nguyên thủy bậc n). Ta định nghĩa **n-th root of unity** (căn đơn vị bậc n) là các số phức thoả phương trình $w^n = 1$. Trong đó tính **primitive** (nguyên thủy) cho biết các số phức w được coi là nghiệm khi và chỉ khi nó không phải là k-th root of unity với bất kì $k < n$.

Ta có 2 tính chất quan trọng sau [1]:

- **Reflection** (Phản xạ): Nếu w là primitive n-th root of unity, trong đó $n \geq 2$ và n chẵn, thì $w^{k+n/2} = -w^k$ với mọi $k = 0, 1, \ldots, n/2 - 1$.
- **Reduction** (Phân rã): Nếu w là primitive (2n)-th root of unity, thì w^2 là primitive n-th root of unity.

Từ 2 tính chất trên, đặt $x = w^k$, ta có thể viết lại đa thức $P(x)$ thành:

$$P(w^k) = P_{even}(w^{2k}) + w^kP_{odd}(w^{2k})$$
$$\implies P(w^{k+n/2}) = P_{even}(w^{2k}) - w^{k+n/2}P_{odd}(w^{2k})$$

Từ đây, ta cũng viết lại đa thức $y = P(x)$ thành:

$$y_i = \sum_{j=0}^{n-1} a_j \times w^{ij} \quad \text{với } i = 0, 1, \ldots, n-1$$

5.1.1.3 Schönhage–Strassen

Cho 2 số nguyên viết dưới dạng đa thức là $A(x)$ và $B(x)$, khi đó [2]:

$$(A \times B)(x) = A(x) \times B(x)$$
$$\implies \text{FFT}(A \times B) = \text{FFT}(A) \times \text{FFT}(B)$$
$$\implies A \times B = \text{FFT}^{-1}(\text{FFT}(A) \times \text{FFT}(B))$$

Ta sẽ gọi FFT^{-1} là **Inverse Fast Fourier Transform**, có thuật toán tương tự như FFT, tuy nhiên FFT^{-1} sẽ có tác dụng chuyển đổi từ một vector các giá trị y về một coefficient vector a.

[1] Các chứng minh sẽ không được đề cập để giản lược nội dung của báo cáo
[2] Các chứng minh sẽ không được đề cập để giản lược nội dung của báo cáo

Hay nói cách khác FFT^{-1} giống như một phép **interpolation** (nội suy) tìm một đa thức thoả mãn các giá trị cho trước, bằng một vài phép biến đổi ma trận, ta sẽ có được công thức bên dưới [3]:

$$a_i = \frac{1}{n} \sum_{j=0}^{n-1} y_j \times w^{-ij} \quad \text{với } i = 0, 1, \dots, n-1$$

Tất nhiên độ dài vector giá trị y và độ dài coefficient vector a là bằng nhau và bằng n. Vì với n phần tử là đủ để xác định một đa thức bậc $n-1$. Đây là tính chất cơ bản trong phép interpolation.

Sau khi có được coefficient vector $A \times B$, ta sẽ thực hiện một số phép toán như **normalize** (chuẩn hoá) hay **carry and reduce** (nhớ và rút gọn) để có được kết quả cuối cùng.

[3] Các chứng minh sẽ không được đề cập để giản lược nội dung của báo cáo

5.1.1.4 Fast Fourier Transform pseudocode

Algorithm: Fast Fourier Transform (FFT)
Input:
An n-length coefficient vector $a \leftarrow [a_0, a_1, \dots, a_{n-1}]$
If FFT is used for inverse transform, set $invert \leftarrow true$
Procedure: FFT(a, invert)
if $n \leftarrow 1$ **then**
return $y \leftarrow a$
end if
divide step
$a_{\text{even}} \leftarrow [a_0, a_2, a_4, \dots, a_{n-2}]$
$a_{\text{odd}} \leftarrow [a_1, a_3, a_5, \dots, a_{n-1}]$
recursive calls with w^2 as $(n/2)$-th root of unity
FFT(a_{even}, invert)
FFT(a_{odd}, invert)
combine step, using $x \leftarrow w^i$
if invert **then**
angle $\leftarrow -2 \times \pi/n$
else
angle $\leftarrow 2 \times \pi/n$
$w \leftarrow 1,\ \ w_n \leftarrow \cos(\text{angle}) + i \times \sin(\text{angle})$
for $i \leftarrow 0, 1, \dots, n/2 - 1$ **do**
$y[i] \leftarrow a_{\text{even}}[i] + w \times a_{\text{odd}}[i]$
$y[i + n/2] \leftarrow a_{\text{even}}[i] - w \times a_{\text{odd}}[i]$
if invert **then**
$y[i] \leftarrow y[i]/2$
$y[i + n/2] \leftarrow y[i + n/2]/2$
end if
$w \leftarrow w \times w_n$
end for
End Procedure

Sử dụng **Master Theorem** (Định lý Master), ta nhận thấy thuật toán chia ra làm 2 phần, mỗi phần có kích thước là $n/2$ còn lại có độ phức tạp là $O(n)$, do đó độ phức tạp thời gian của thuật toán là:

$$T(n) = 2 \times T\left(\frac{n}{2}\right) + O(n) = O(n \log(n))$$

5.1.1.5 Schönhage–Strassen pseudocode

Algorithm: Schönhage–Strassen algorithm for big integers multiplication
Input:
A left-hand side coefficient vector $a \leftarrow [a_0, a_1, \dots, a_{n-1}]$
A right-hand side coefficient vector $b \leftarrow [b_0, b_1, \dots, b_{n-1}]$
Output:
A coefficient vector $c \leftarrow [c_0, c_1, \dots, c_{2n-1}]$ where $c = a \times b$
Function: Schönhage–Strassen(a, b)
$n \leftarrow 1$
split and zero-pad step
while $n < \max(a.\text{size}(), b.\text{size}())$ **do**
$n \leftarrow n \times 2$
end while
a.resize(n), b.resize(n)
apply FFTs
FFT(a), FFT(b)
$c \leftarrow a \times b$
FFT$^{-1}(c)$
perform carry and reduce
recarry(c)
End Function

Với mỗi FFT ta có độ phức tạp là $O(n\log(n))$, do đó độ phức tạp thời gian của thuật toán là:

$$T(n) = n \times T\left(2n + \frac{1}{2}\log(n)\right) + O(n\log(n)) = O(n\log(n)\log(\log(n)))$$

5.1.2 Right-to-left Binary algorithm: $A^{\mathbf{exp}} \mod n$

Modular Exponentiation (Lũy thừa theo Modulo) cũng là một trong những phép toán quan trọng trong các bài toán **Public Key Cryptography** (Mã hoá Khoá Công khai) như RSA, ElGamal, Diffie-Hellman,...

Phép modular exponentiation được biểu diễn: $A^{\text{exp}} \mod n$.

Trong trường hợp A là một số nguyên dương lớn, việc vừa tính A^{exp} vừa lấy modulo n sẽ rất tốn thời gian với độ phức tạp là $O(\text{exp})$. Thay vào đó, chúng ta có một số thuật toán để thực hiện phép tính này với độ phức tạp nhỏ hơn nhiều là $O(\log(\text{exp}))$.

5.1.2.1 Right-to-left Binary

Ý tưởng của thuật toán này là chuyển đổi số mũ exp sang hệ nhị phân:

$$\text{exp}_2 = \sum_{i=0}^{k-1} a_i \times 2^i \quad \text{với } a_i \in \{0, 1\}$$

Trong đó:

- $a_i \in \{0, 1\}$ là bit thứ i của $\exp_2$.
- k là số bit của $\exp_2$.
- $\exp_2 = \overline{a_{k-1}a_{k-2}...a_1a_0}$ là biểu diễn nhị phân của exp.

Khi đó, giá trị của $A^{\exp}$ có thể được viết lại thành:

$$\begin{aligned} A^{\exp} &= A^{\left(\sum_{i=0}^{k-1} a_i \times 2^i\right)} \\ &= \prod_{i=0}^{k-1} A^{a_i \times 2^i} \end{aligned}$$

Bằng cách này, thay vì lặp exp lần, ta chỉ cần lặp k lần, nhanh hơn rất nhiều.

5.1.2.2 Right-to-left Binary pseudocode

Algorithm: Right-to-left Binary
Input:
A base A
An exponent exp
A modulo n
Function: Right-to-left Binary$(A, \exp, n)$
if $n = 1$ **then**
return 0
end if
result $\leftarrow 1$
$A \leftarrow A \pmod n$
while exp > 0 **do**
if exp $= 1 \pmod 2$ **then**
result $\leftarrow$ result $\times A \pmod n$
$A \leftarrow A \times A \pmod n$
exp $\leftarrow$ exp $>> 1$
end while
return result
End Function

Ta nhận thấy trong vòng lặp `while`, số lần lặp tuân theo điều kiện exp > 0 và mỗi lần lặp, exp dịch bit sang phải 1 (tương đương với việc chia cho 2). Hay nói cách khác, exp liên tục phân rã với hệ số 2, do đó số lần lặp của vòng lặp `while` tuân theo hàm log là $O(\log(\exp))$.

5.1.3 Others [4]

Ngoài ra, còn có một số thuật toán khác đơn giản hơn như:

- **Bisection method** cho $\sqrt{A}$
- **Extended Euclidean algorithm** cho $A^{-1} \pmod n$
- **Euclidean algorithm** cho $\gcd(A, B)$
- ...

[4] Chi tiết các thuật toán sẽ không được giới thiệu để giản lược nội dung của báo cáo

5.2 Primality test

Kiểm tra tính số nguyên tố là một lĩnh vực lớn trong toán học, tính đến nay đã có vô số các phương pháp để kiểm tra tính số nguyên tố của một số nguyên dương A. Trong đó có một phương pháp kiểm (thử) tra số nguyên tố bằng xác suất, đánh đổi độ chính xác với độ phức tạp. So với các số nguyên nhỏ, các số nguyên tương đối lớn không có sự lựa chọn giữa các phương pháp nào khác ngoài phương pháp này.

5.2.1 Fermat's little theorem

Theo **Fermat's little theorem** (định lý Fermat nhỏ), với mọi số nguyên tố p và mọi số nguyên a không chia hết cho n, ta có:

$$\begin{aligned} a^{n-1} &\equiv 1 \pmod{n} \\ \implies a^{n-1} - 1 &\equiv 0 \pmod{n} \end{aligned}$$

Giả sử n là một số nguyên tố, với $n \geq 3$. Khi đó, ta có thể biết rằng $n-1$ là một số chẵn, do đó ta có thể viết lại $n-1$ thành:

$$n - 1 = 2^s \times d$$

Trong đó:

- s là một số nguyên dương bất kì.
- d là một số nguyên dương lẻ.

Đặt a là một số nguyên dương bất kì, $a \in [1, n-1]$. Ta có thể viết lại thành:

$$\begin{aligned} a^{n-1} - 1 &\equiv 0 \pmod{n} \\ \implies a^{2^s \times d} - 1 &\equiv 0 \pmod{n} \\ \implies \left(a^{2^{s-1} \times d}\right)^2 - 1 &\equiv 0 \pmod{n} \\ \implies \left(a^{2^{s-1} \times d} - 1\right)\left(a^{2^{s-1} \times d} + 1\right) &\equiv 0 \pmod{n} \\ \implies \left(\left(a^{2^{s-2} \times d}\right)^2 - 1\right)\left(a^{2^{s-1} \times d} + 1\right) &\equiv 0 \pmod{n} \\ \implies \left(a^{2^{s-2} \times d} - 1\right)\left(a^{2^{s-2} \times d} + 1\right)\left(a^{2^{s-1} \times d} + 1\right) &\equiv 0 \pmod{n} \\ &\dots \\ \implies \left(a^d - 1\right)\left(a^d + 1\right)\left(a^{2d} + 1\right) \dots \left(a^{2^{s-1} \times d} + 1\right) &\equiv 0 \pmod{n} \end{aligned}$$

Điều này có nghĩa là khi n là một số nguyên tố, một trong các nhân tử phía trên sẽ phải chia hết cho n. Do đó:

$$a^d \equiv 1 \pmod{n} \quad \vee \quad a^{2^i \times d} \equiv 1 \pmod{n} \quad \text{với } i = 0, 1, \dots, s-1$$

Trong trường hợp này, ta nói rằng a là một **Miller-Rabin non-witness** (không phải nhân chứng Miller-Rabin) của n (hay n là một **strong pseudoprime** (giả nguyên tố mạnh) với cơ số a).

5.2.2 Miller-Rabin algorithm

Tương tự với ví dụ trên, giả sử n là một số nguyên tố, với $n \geq 3$. Chọn một số nguyên dương a bất kì, $a \in [1, n-1]$. Ta có thể viết lại thành:

Ta nói rằng a là một **Miller-Rabin witness** (nhân chứng Miller-Rabin) của n nếu:

$$a^d \not\equiv 1 \pmod{n} \quad \wedge \quad a^{2^i \times d} \not\equiv -1 \pmod{n} \quad \text{với } i = 0, 1, \ldots, s-1$$

Cụm từ "witness" (nhân chứng) có nghĩa là a là một nhân chứng để chứng minh rằng n là một hợp số. Do đó nếu n không có Miller-Rabin witness nào, ta có thể kết luận rằng n là một số nguyên tố (trừ trường hợp $n = 2$).

Thuật toán Miller-Rabin sẽ chọn ngẫu nhiên một số a trong khoảng $[1, n-1]$ và kiểm tra biểu thức trên với mọi $i = 0, 1, \ldots, s-1$. Nếu tất cả các biểu thức đều thoả mãn, ta có thể kết luận rằng n là một số nguyên tố với một xác suất sai là $\frac{1}{4}$.

Do đó, chỉ cần lặp k lần thuật toán Miller-Rabin, xác suất sai sẽ giảm xuống chỉ còn 4^{-k}.

5.2.3 Miller-Rabin pseudocode

Algorithm: Miller-Rabin
Input:
A tested number n
A number of iterations k
Function: Miller-Rabin(n, k)
if $n = 2$ **then**
return true
end if
if $n = 1$ **or** n is even **then**
return false
end if
$n - 1 \leftarrow 2^s \times d$
for $i \leftarrow 0, 1, ..., k - 1$ **do**
$a \leftarrow \text{random}(1, n - 1)$
$x \leftarrow a^d \pmod{n}$
if $x = 1$ **then**
continue
end if
for $j \leftarrow 0, 1, ..., s - 1$ **do**
if $x = n - 1$ **then**
continue
end if
$x \leftarrow x^2 \pmod{n}$
end for
return false
end for
return true
End Function

Bằng một số phương pháp xác suất và thống kê, ta chứng minh được độ phức tạp thời gian của thuật toán trên là $O(k \log^3(n))$.

5.3 RSA - Key Generation

Gọi e là **public exponent** (số mũ công khai), d là **secret exponent** (số mũ bí mật).

Cho trước 2 số nguyên tố p, q. Khi đó với một số e là **public exponent** (số mũ công khai), ta có thể tìm được một số d là **secret exponent** (số mũ bí mật) bằng cách:

$$\begin{aligned} & d \equiv e^{-1} \pmod{\varphi} \\ \implies & d \equiv e^{-1} \pmod{(p-1)(q-1)} \end{aligned}$$

5.4 RSA - Decryption and Encryption

Cho trước một public key (N, e), ta có thể mã hoá một số nguyên m là tin nhắn cần gửi đến địa chỉ (N, e) bằng cách:

$$c \equiv m^e \pmod{N}$$

Tương tự, ta có thể dùng secret key (N, d) để giải mã bản mã c bằng cách:

$$m \equiv c^d \pmod{N}$$

Chứng minh rất đơn giản, theo **Fermat's little theorem** (định lý Fermat nhỏ), ta có:

$$\begin{aligned} ed &\equiv 1 \pmod{p-1} \\ ed &\equiv 1 \pmod{q-1} \end{aligned}$$

Vì p và q là 2 số nguyên tố cùng nhau, nên theo **Chinese remainder theorem** (định lý số dư Trung Hoa), ta có:

$$\begin{aligned} & ed \equiv 1 \pmod{(p-1)(q-1)} \\ \implies & m \equiv m^{ed} \pmod{N} \\ \implies & m \equiv (m^e)^d \pmod{N} \\ \implies & m \equiv c^d \pmod{N} \end{aligned}$$

6 Disclaimer

Toàn bộ nội dung của báo cáo này đều được research trong thời gian ngắn, do đó không thể tránh khỏi sai sót.

Tuy nhiên tất cả nội dung được ghi trên báo cáo đã được bản thân em chứng minh, hiểu rõ và nắm chắc. Do đó bất kì sai sót nào trong báo cáo này đều là do sai sót của chính bản thân em, không phải do copy hay cắt ghép từ các nguồn lại với nhau.

Với những sai sót nghiêm trọng, xin vui lòng liên hệ với em qua email: nguyenhuuthuan25112003@gmail.com. Em xin cảm ơn.

7 Final words

Em là Nguyễn Hữu Thuận, 3 năm học tại HCMUS, 2 năm tiếp xúc với Blockchain và 1 năm kinh nghiệm làm việc Blockchain Developer. Tuy vậy, đến gần đây em mới hứng thú với lĩnh vực Mã hoá - Mật mã, đặc biệt là các ứng dụng của nó và Zero Knowledge.

Vì sự chủ quan về kiến thức của bản thân, không để ý đến % điểm thành phần, bận rộn trong cuộc sống và một số tai nạn ngoài ý muốn. Em đã bỏ lỡ các bài tập trên lớp và về nhà, thứ chiếm đến $60\% \times 70\% = 42\%$ điểm tổng kết môn học. Điều này khiến em cảm thấy rất hối hận và tiếc nuối vì nguy cơ phải học lại. Hơn nữa, đây lại là môn học và em yêu thích nhất.

Hi vọng bài báo cáo này có thể giúp em có thể đạt được điểm số mong muốn để qua môn, em xin cảm ơn các thầy.