



BÁO CÁO LAB 3

Public-key Cryptography - RSA & Diffie-Hellman Key Exchange *Môn Mật mã học*

Sinh viên thực hiện	Sinh viên MSSV: 16520596 Họ tên: Đinh Hồ Anh Khoa
Lớp	NT119.I21.ANTN
Tổng thời gian thực hiện Lab trung bình	
Ý kiến (nếu có) + Khó khăn + Đề xuất, góp ý...	Em trễ vài phút ạ, em xin lỗi ạ
Điểm tự đánh giá (bắt buộc)	9.5 /10



Nội dung

Public-key Cryptography - RSA & Diffie-Hellman Key Exchange	1
1. Task 3.1	3
1.1. Số nguyên tố	3
1.2. Xác định Ước chung lớn nhất (gcd) của 2 số lớn nhất sinh viên có thể xử lý. 10	
1.3. Tính giá trị của phép lũy thừa modulo $a^x \bmod p$ có thể áp dụng cho các trường hợp lũy thừa lớn (>40), ví dụ $7^{40} \bmod 19$	11
2. Task 3.2	12
2.1. Xác định khóa công khai PU và khóa riêng PR	12
2.2. Sử dụng khóa trong trường hợp $p1; q1; e1$ trên để thực hiện mã hóa và giải mã thông điệp M=5 trong 2 trường hợp mã hóa bảo mật và mã hóa chứng thực	13
2.3. Sử dụng lần lượt các khóa trên để thực hiện mã hóa thông điệp sau: The University of Information Technology Xác định Ciphertext tương ứng dưới dạng Base64	14
2.4. Xác định Plaintext tương ứng của các Ciphertext sau, biết rằng chúng được mã hóa bởi 1 trong 2 bộ khóa trên	16
3. Mở rộng 3.1	18
Tại sao dùng số random trong mật mã:	19
RSA có phải mã hóa khối không	20
2048 bit biểu thị được số thập phân bao nhiêu chữ số thập phân	20
4. Task 3.3	21
5. Task 3.4	27
Trong giao thức trao đổi khóa Diffie-Hellman (DH), nếu kẻ tấn công có các giá trị g ; p ; $g^a \bmod p$; $g^b \bmod p$ thì khóa bí mật đang trao đổi có bị lộ không? Tại sao?	27
DH có hoàn toàn an toàn không? Mô tả hình thức tấn công và cách hạn chế (nếu có)	27
6. Tài liệu tham khảo	28



1.Task 3.1

1.1. Số nguyên tố

Phát sinh số nguyên tố ngẫu nhiên có độ dài 8 bit, 32 bit, 64 bit:

Giải: Khởi tạo mảng byte có độ dài sao cho đúng với số bit yêu cầu, sau đó set giá trị cho mảng byte này một cách ngẫu nhiên rồi chuyển mảng byte sang kiểu dữ liệu BigInteger. Dùng thuật toán Miller Rabin để kiểm tra số đó có phải số nguyên tố không, nếu không thì giảm dần cho tới khi số đó là số nguyên tố.

8bit:

The screenshot shows a Java Swing window titled "Form1". Inside the window, there is a text input field containing the number "8". To its right is a list box containing the number "113". Further to the right is a dark grey button labeled "Generate". The window has standard Windows-style title bar controls (minimize, maximize, close) in the top right corner.

32bit:

The screenshot shows a Java Swing window titled "Form1". Inside the window, there is a text input field containing the number "32". To its right is a list box containing the number "102374917". Further to the right is a dark grey button labeled "Generate". The window has standard Windows-style title bar controls (minimize, maximize, close) in the top right corner.



64bit:

The screenshot shows a Java Swing window titled "Form1". Inside the window, there is a text input field containing the number "64". To the right of this field is a large text area containing the hexadecimal string "899639095594407769". To the right of the text area is a button labeled "Generate". The window has standard Windows-style window controls (minimize, maximize, close) in the top right corner.

Kiểu BigInteger có thể biểu diễn chữ số rất lớn dựa vào dung lượng bộ nhớ của máy nên việc tạo số nguyên tố 1024bit hay nhiều hơn nữa vẫn thực hiện được (sẽ nói rõ hơn về độ lớn của BigInteger ở phần sau):

The screenshot shows a Java Swing window titled "Form1". Inside the window, there is a text input field containing the number "1028". To the right of this field is a large text area containing a long hexadecimal string: "10565660171110143378176109225285125509514695290003104537313325512218030167782436180628132584701285492209516692110774062177817967215095699025743775729186652486976454732985282565572570001040936169614991750918024953009703065663834424334061207366058028625926240153990769687092508957328450020967279295369269148357". To the right of the text area is a button labeled "Generate". The window has standard Windows-style window controls (minimize, maximize, close) in the top right corner.



Xác định 10 số nguyên tố lớn nhất nhỏ hơn 10 số nguyên tố Mersenne đầu tiên:

Giải: lưu 10 số nguyên tố đầu tiên và thực hiện giảm dần cho tới khi gặp số nguyên tố.
kiểm tra nguyên tố bằng Miller Rabin

10 số nguyên tố Mersenne đầu tiên:

#	p	M_p	M_p digits	Discovered	Discoverer	Method used
1	2	3	1	c. 430 BC	Ancient Greek mathematicians ^[19]	
2	3	7	1	c. 430 BC	Ancient Greek mathematicians ^[19]	
3	5	31	2	c. 300 BC	Ancient Greek mathematicians ^[20]	
4	7	127	3	c. 300 BC	Ancient Greek mathematicians ^[20]	
5	13	8191	4	1456	Anonymous ^{[21][22]}	Trial division
6	17	131071	6	1588 ^[23]	Pietro Cataldi	Trial division ^[24]
7	19	524287	6	1588	Pietro Cataldi	Trial division ^[25]
8	31	2147483647	10	1772	Leonhard Euler ^{[26][27]}	Enhanced trial division ^[28]
9	61	2305843009213693951	19	1883 November ^[29]	Ivan M. Pervushin	Lucas sequences
10	89	618970019642...137449562111	27	1911 June ^[30]	Ralph Ernest Powers	Lucas sequences

Số thứ nhất:

Form1

(1)First Find

2



Số thứ hai:

Form1

(2)Second Find

5

Số thứ 3:

Form1

(3)Third Find

23

Số thứ 4:

Form1

(4)Fourth Find

113



Số thứ 5:

Form1

(5)Fifth Find

8179

Số thứ 6:

Form1

(6)Sixth Find

131063

Số thứ 7:

Form1

(7)Seventh Find

524269



Số thứ 8:

The screenshot shows a web browser window with a title bar containing minimize, maximize, and close buttons. The page title is "Form1". Below the title, there is a dropdown menu with the text "(8)Eighth" and a downward arrow. To the right of the dropdown is a dark grey button labeled "Find". Below these elements is a large rectangular text area containing the number "2147483629". In the bottom right corner of the page, there are three small dots arranged in a triangle.

Số thứ 9:

The screenshot shows a web browser window with a title bar containing minimize, maximize, and close buttons. The page title is "Form1". Below the title, there is a dropdown menu with the text "(9)Nineth" and a downward arrow. To the right of the dropdown is a dark grey button labeled "Find". Below these elements is a large rectangular text area containing the number "2305843009213693921". In the bottom right corner of the page, there are three small dots arranged in a triangle.

Số thứ 10:

The screenshot shows a web browser window with a title bar containing minimize, maximize, and close buttons. The page title is "Form1". Below the title, there is a dropdown menu with the text "(10)Tenth" and a downward arrow. To the right of the dropdown is a dark grey button labeled "Find". Below these elements is a large rectangular text area containing the number "618970019642690137449562091". In the bottom right corner of the page, there are three small dots arranged in a triangle.



Kiểm tra một số nguyên bất kỳ nhỏ hơn $2^{89}-1$ có phải số nguyên tố không.

Giải: sử dụng thuật toán Miller Rabin trên kiểu BigInteger

Nhập một dãy số bất kỳ từ bàn phím (và tất nhiên rất khó để nó là số nguyên tố):

Form1

13265779803246789814325367586675644356554656645365434215

Check

13265779803246789814325367586675644356554656645365434215 IS NOT Prime

Sử dụng chương trình tạo số nguyên tố ngẫu nhiên ở trên và kiểm tra (không thể dùng tool online để kiểm được vì đa số đều giới hạn input):

Form1

96

16264439863093901135043574871

Generate

Form1

16264439863093901135043574871

Check

16264439863093901135043574871 IS Prime



1.2. Xác định Ước chung lớn nhất (gcd) của 2 số lớn nhất sinh viên có thể xử lý.

Giải: Sử dụng thuật toán Euclid để tìm gcd của hai số kiểu BigInteger

Sử dụng hai số random có độ dài 80bit (80 bit có thể chứa được chữ số gồm 50 ký tự):

The screenshot shows a Java Swing window titled "Form1" with three input fields and two buttons. The first input field is labeled "First Number:" and contains the value "323504816793242008063438642776112597377721804347". Below it is a "Generate" button. The second input field is labeled "Second Number:" and contains the value "646474465782010307812716202483103121255983443661". Below it is another "Generate" button. The third input field is labeled "Result:" and contains the value "257". Below it is a "Calculate" button.

Thực chất có thể tìm gcd của hai số có độ dài lớn hơn rất nhiều(trong hình đã xuống dòng khá nhiều):

The screenshot shows a Java Swing window titled "Form1" with three input fields and two buttons. The first input field is labeled "First Number:" and contains a long number with line breaks: "45685643214567890765467865431245621346579876435674532365751876756441" and "23165365981923847521341873465901283740918237583274659812736492104378". Below it is a "Generate" button. The second input field is labeled "Second Number:" and contains a long number with line breaks: "12347238907539012746324986792345816253748653498197823642139485743658" and "37129837461897561234123578652314871523487216349872159134586876123438". Below it is another "Generate" button. The third input field is labeled "Result:" and contains the value "2". Below it is a "Calculate" button.



BigInteger sử dụng mảng các byte để lưu trữ nên có một chữ BigInteger có thể biểu diễn một số có dung lượng 2GB (tương đương khoảng 600 triệu chữ số thập phân):

Memory Limit

4

BigInteger relies on int array for storage. Assuming this, theoretical limit for maximum number, that BigInteger capable to represent, can be derived from maximum array size available in .net. There is a SO topic about arrays here: [Finding how much memory I can allocate for an array in C#](#).

Assuming that we know maximum array size, we can estimate maximum number, which BigInteger can represent: $(2^{32})^{\text{max_array_size}}$, where:

- 2^{32} - maximum number in array cell (int)
- max_array_size - maximum allowed size of int array which is limited by object size of 2GB

This gives number with 600 millions of decimal digits.

Performance Limit

As for performance, BigInteger uses [Karatsuba algorithm](#) for multiplication and linear algorithm for adding. Multiplication complexity is $3n^{\log_2 3} \approx 3n^{1.585}$, that means it will scale pretty well even for large numbers ([Complexity graph](#)), however you can still hit performance penalty depending on size of RAM and processor cache.

As far, as maximum number size is limited to 2GB, on descent machine you won't see unexpected performance gap, but still operating on 600 million digit numbers will be dead slow.

Nếu tài nguyên máy tính đủ lớn và có đủ thời gian để chờ thì vẫn có thể xử lý được hai số có khoảng 600 triệu chữ số thập phân

1.3. Tính giá trị của phép lũy thừa modulo $a^x \bmod p$ có thể áp dụng cho các trường hợp lũy thừa lớn (>40), ví dụ $7^{40} \bmod 19$

Giải: Kiểu BigInteger có hỗ trợ thực hiện phép tính này



Form1

$a^x \bmod p$

Calculate

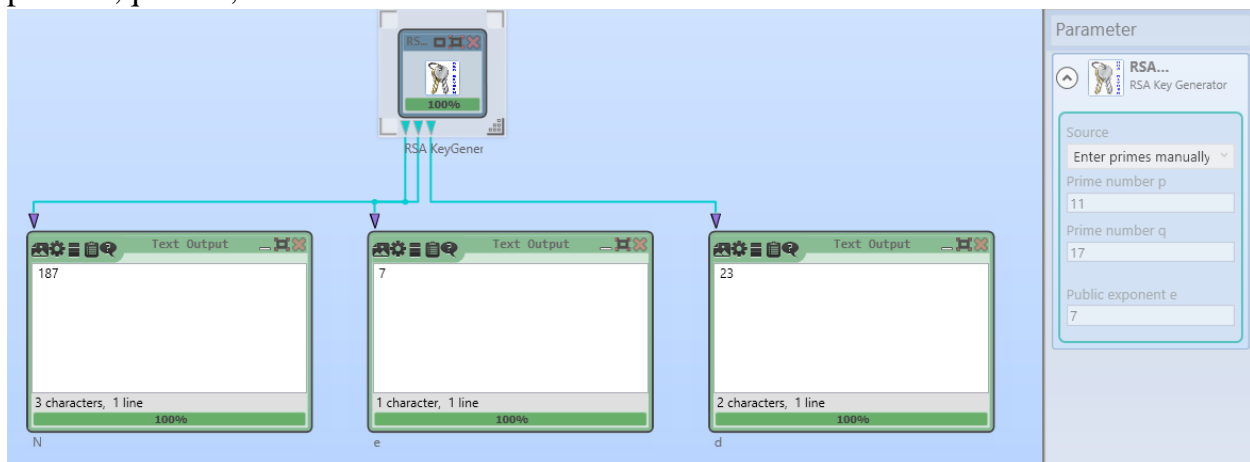
Number a: 123 Number x: 456 Number p: 789

Result: 699

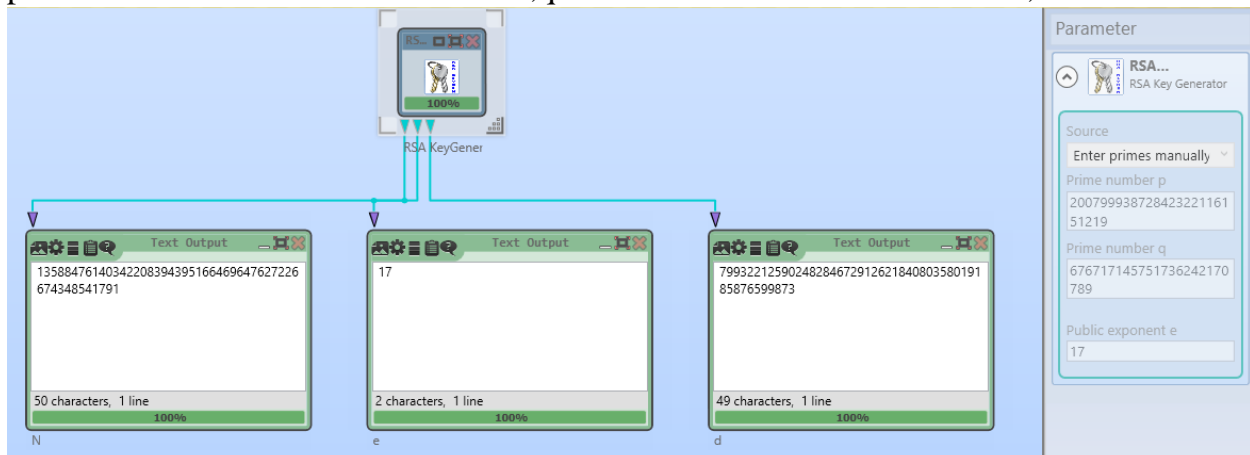
2. Task 3.2

2.1. Xác định khóa công khai PU và khóa riêng PR

$p_1 = 11; q_1 = 17; e_1 = 7$

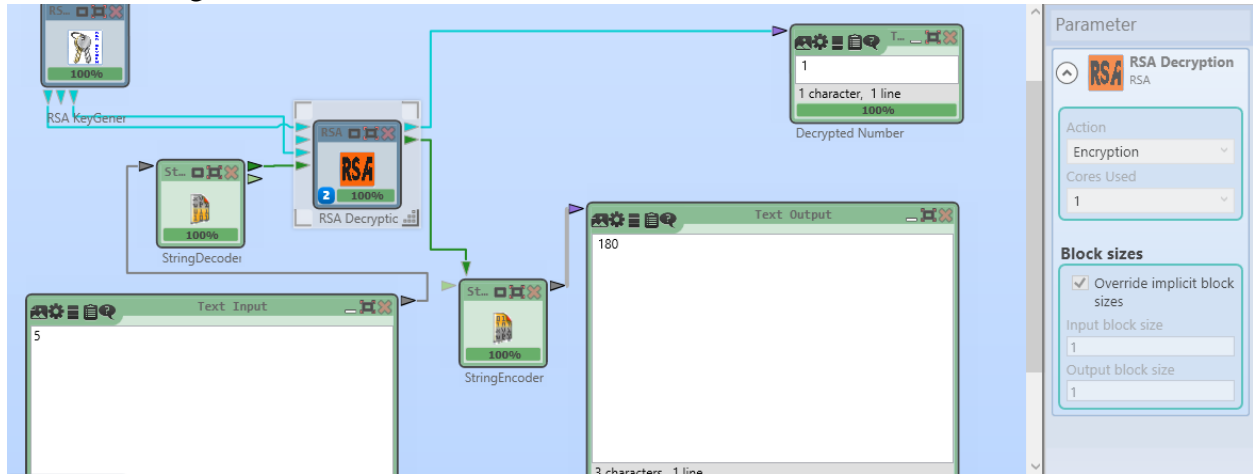


$p_2 = 20079993872842322116151219; q_2 = 676717145751736242170789; e_2 = 17$

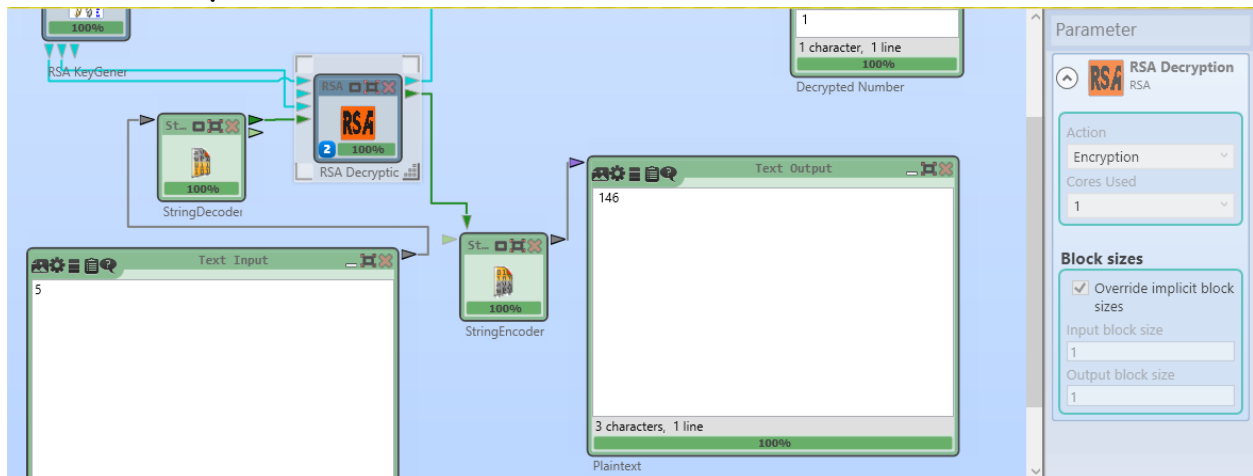


2.2. Sử dụng khóa trong trường hợp $p_1; q_1; e_1$ trên để thực hiện mã hóa và giải mã thông điệp $M=5$ trong 2 trường hợp *mã hóa bảo mật* và *mã hóa chứng thực*.

Mã hóa chứng thực:



Mã hóa bảo mật:

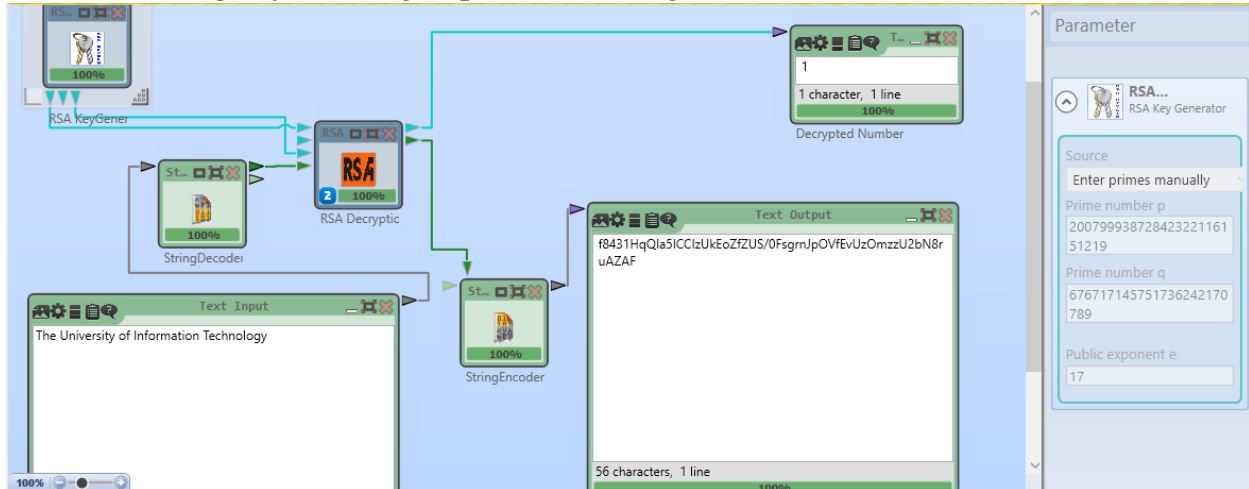


2.3. Sử dụng lần lượt các khóa trên để thực hiện mã hóa thông điệp sau:

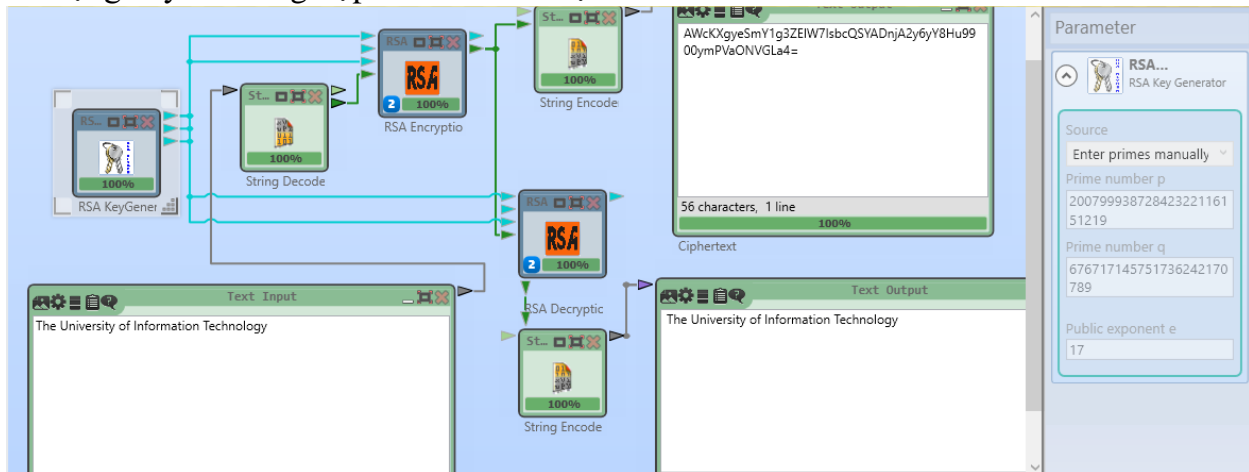
The University of Information Technology

Xác định Ciphertext tương ứng dưới dạng Base64

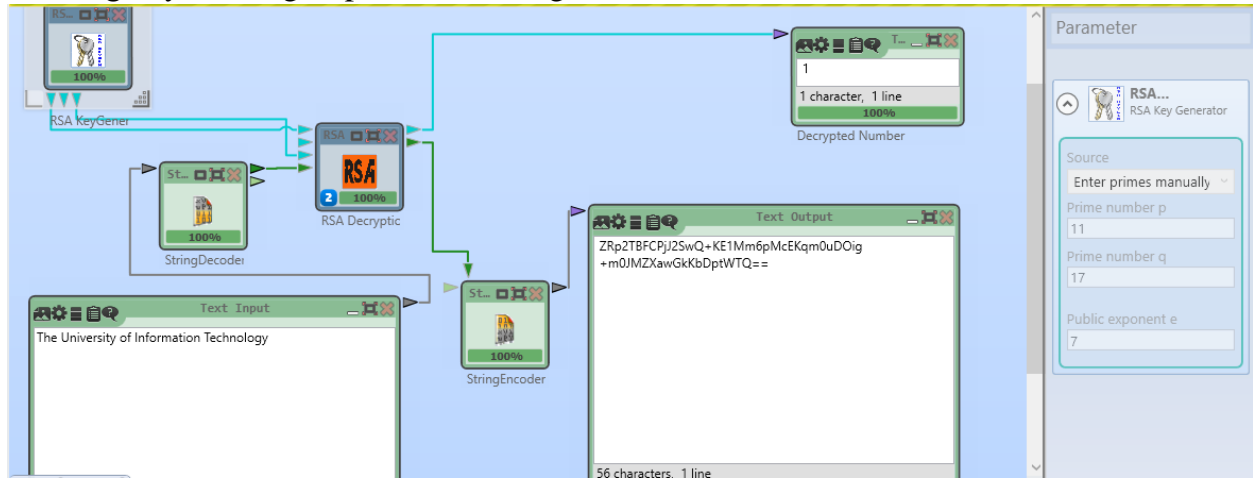
Mã hóa sử dụng key 2 trường hợp mã hóa chứng thực:



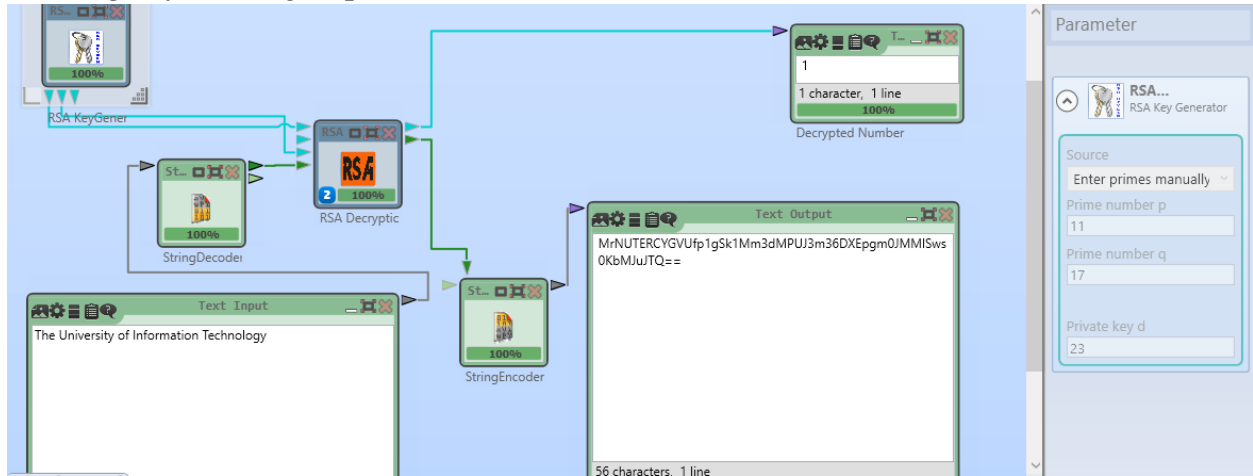
Sử dụng key 2 trường hợp mã hóa bảo mật:



Sử dụng key1 trường hợp mã hóa chứng thực



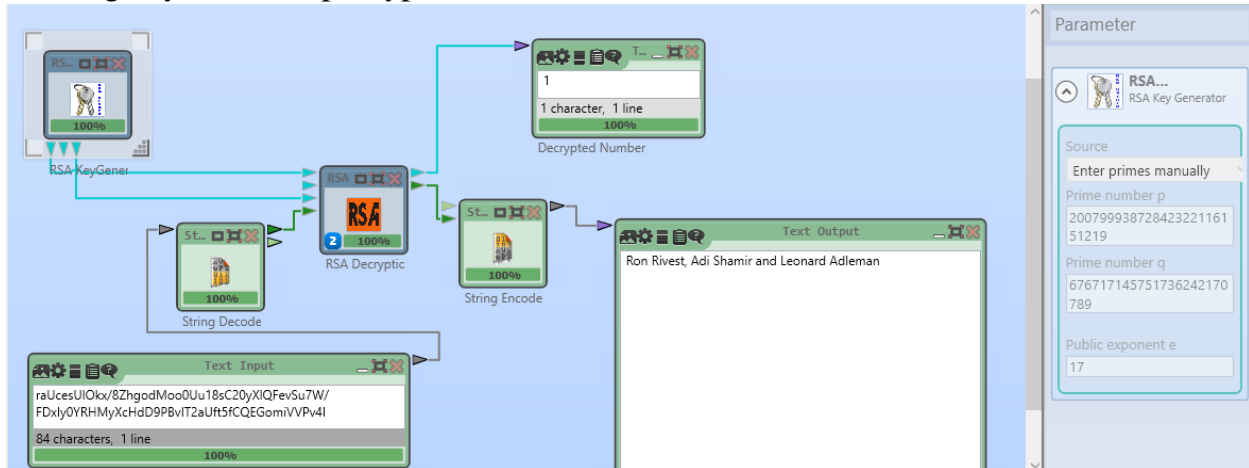
Sử dụng key1 trường hợp mã hóa bảo mật



2.4. Xác định Plaintext tương ứng của các Ciphertext sau, biết rằng chúng được mã hóa bởi 1 trong 2 bộ khóa trên

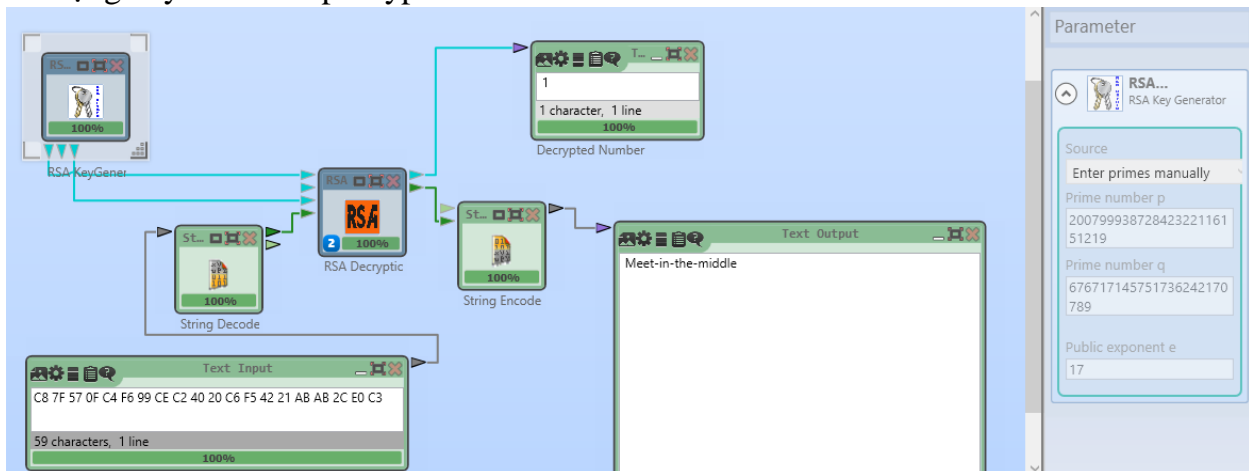
(a) raUcesUIOkx/8ZhgodMoo0Uu18sC20yXlQFevSu7W/FDxly0YRHMMyXcHdD9PBvIT2aUft5fCQEGomiVVPv4I

Sử dụng key thứ hai input type là base64



(b) C8 7F 57 0F C4 F6 99 CE C2 40 20 C6 F5 42 21 AB AB 2C E0 C3

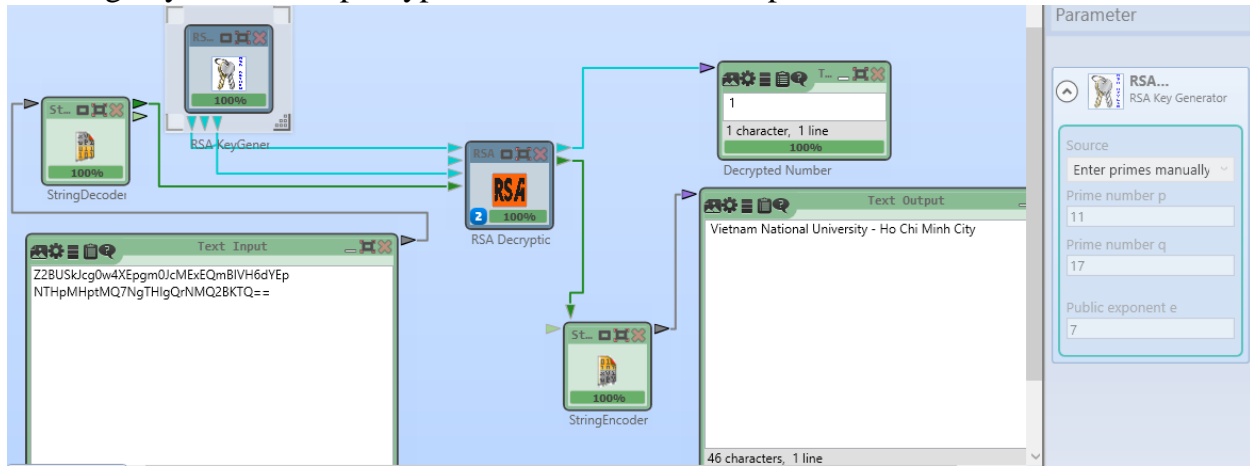
Sử dụng key thứ hai input type hexadecimal:





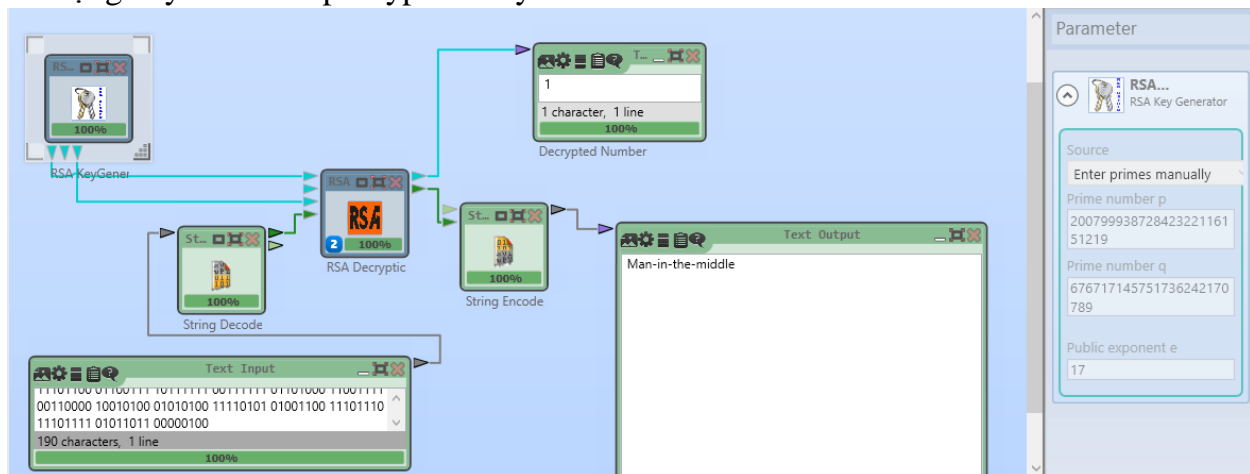
(c) Z2BUSkJcg0w4XEpgm0JcMExEQmBIVH6dYEpNTHpMHptMQ7NgTHlgQrNMQ2BKTQ==

Sử dụng key thứ nhất input type base64 có override implicit block sizes



(d) 00101000 00010100 11111111 10110111 00101110 11001010
11101100 01100111 10111111 00111111 01101000 11001111 00110000
10010100 01010100 11110101 01001100 11101110 11101111 01011011 00000100

Sử dụng key thứ hai input type binary





3. Mở rộng 3.1

Sử dụng lại hàm sinh số nguyên tố ngẫu nhiên, và plugin của cryptool

Bấm vào nút Generate để tạo số p và q ngẫu nhiên độ dài 1024bit:

Sau đó chọn kiểu mã hóa, giải mã và điền nội dung cần mã hóa, giải mã.



Tại sao dùng số random trong mật mã:

Trong một vài trường hợp ta chỉ cần các số người khác không thể đoán được là đủ (initialization vectors cho block cipher, key-exchange, chữ ký điện tử,...) hoặc các số ngẫu nhiên để chống lại kiểu tấn công bằng cách thống kê (như monoalphabetic, các mã hóa khối, thông điệp lặp lại nhiều lần) khiến tần số xuất hiện không còn ý nghĩa quan trọng nữa.



For some types of algorithms (or protocols) we only need non-guessable (by the attacker) bits/numbers, not reproducible non-guessable ones (like from a deterministic PRNG).

14



In this cases, "real" random numbers are in theory (i.e. from an information-theoretic point of view, not a cryptographic one) better, since they can't be guessed (or even influenced) by an attacker, even if she could break our PRNG.



Some cases that I can now think of, where we don't need deterministic random numbers:

- key generation (both symmetric and asymmetric)
- initialization vectors for [block cipher modes of operation](#) (these are usually sent with the message, so the same plaintext will not result in recognizable ciphertext for the next message)
- random padding in asymmetric encryption (for example [OAEP](#))
- salts for password storage (these are stored with the hash)
- challenges in zero knowledge proofs (sent to the partner)
- random values used in digital signatures (the `k` in DSA)
- [one time pads](#) (OTP) (Here, for the security proof, we actually *need* "real" random numbers.)
- [chaffing and winnowing](#)

In practice, pseudo-random bits are cheaper and just as secure for real-world attackers (e.g. with resources limited by our earth mass and universe lifetime), as long as the PRNG is not broken and has enough entropy input to start with.

(If the attacker can control the seemingly random input to the "true random" generator, this would be even worse than a good PRNG.)

Often for these uses we use a combination of a cryptographically secure (deterministic) PRNG and an entropy pool, which gets filled (and re-filled) by random bits gathered by the OS. This would be a non-deterministic PRNG.

Một câu trả lời hay trên <https://crypto.stackexchange.com>



RSA có phải mã hóa khối không

RSA không phải mã hóa khối bởi mã hóa khối sử dụng khóa đối xứng (1 khóa dùng để mã hóa và giải mã) nhưng RSA có tới hai khóa, mã hóa bằng khóa này thì giải mã bằng khóa kia, tuy nhiên RSA vẫn có thể sử dụng trong mã hóa khối.


▲ It is neither a stream cipher nor a block cipher. Both of these use the same key to encrypt and decrypt (symmetric encryption).

9 RSA is asymmetric meaning you encrypt with a different key than you decrypt with. The advantage is that the encryption key can be made public, since people can only use it to encrypt and no one can decrypt if you keep the decryption key to yourself.

▼ Unlike (generalization) block and stream ciphers, RSA is based directly on mathematics.

✓ share improve this answer

answered Jan 25 '11 at 15:47

 PulpSpy
2,004 ● 10 ● 18

add a comment

Một câu trả lời trên <https://security.stackexchange.com>

(Đã tham khảo rất nhiều câu trả lời ở nhiều trang nhưng chỉ chụp một vài hình ví dụ)

2048 bit biểu thị được số thập phân bao nhiêu chữ số thập phân

N decimal digits, you need $(98981 * N) / 238370 + 1$ bytes.

=> 2048 bit biểu thị được khoảng 615 chữ số thập phân (đã có áp dụng thử ở 1.2)

▲ Without using floating point arithmetic: For N decimal digits, you need

16 $(98981 * N) / 238370 + 1$


▼ bytes. 98981/238370 is a good rational approximation (from above) to $\log(10)/\log(256)$ (the 9th convergent), integer division truncates, therefore add 1. The formula is tight for $N < 238370$, the number of bytes needed to represent $10^N - 1$ is exactly given by that, it overestimates for N a multiple of 238370 and **really large** N . If you're not too afraid of allocating the odd byte too much, you can also use $(267 * N) / 643 + 1$, $(49 * N) / 118 + 1$, $(5 * N) / 12 + 1$ or, wasting about 10% of space, $(N + 1) / 2$.

✓ As @Henrick Hellström points out, in Delphi one would have to use the `div` operator for integer division (missed the delphi tag).

share improve this answer

edited Feb 26 '12 at 10:13

answered Feb 25 '12 at 11:13

 Daniel Fischer
156k ● 14 ● 252 ● 383

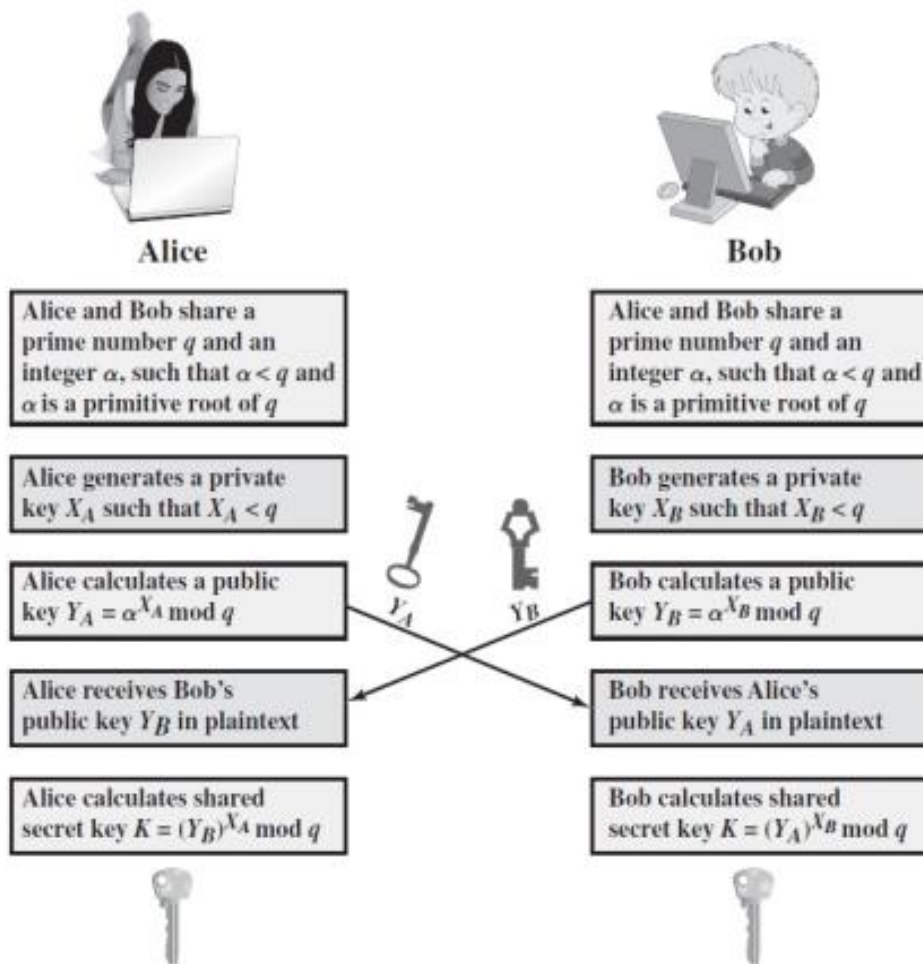
<https://stackoverflow.com/questions/9443236/how-many-bytes-are-required-to-hold-n-decimal-digits>

4.Task 3.3

Giải(xin phép không đi sâu vào thuật toán):

- Xây dựng một ứng dụng chat thông thường giữa client và server.
- Tái sử dụng các thuật toán sinh số nguyên tố, thuật toán Miller Rabin, Modular exponentiation.
- Sau khi chia sẻ khóa có thể dùng khóa đó cho nhiều thuật toán khác nhau, ở đây chỉ sử dụng XOR.
- Ứng dụng chat cho phép gửi tin nhắn mà không cần mã hóa hoặc gửi tin nhắn sau khi mã hóa

Tiến trình chia sẻ khóa:





Đầu tiên khởi chạy server(Bob) sẽ hiện ra IP address:

Bob

Key is: Calculate key

Prime number p: Accept p

Primitive root g: Accpet g

Ciphertext:

Plaintext:

Secret Number b: Send $g^b \text{ mod } p$

Send encrypt Send non-encrypt

Khởi chạy client(Alice) và kết nối đến server

Alice

Key is: Calculate key

Server: 192.168.254.101 Connect

điền ip vào đây sau đó nhấn vào nút connect

Prime number p: Generate

Primitive root g: Calculate

Secret Number a: Send $g^a \text{ mod } p$

Send encrypt Send non-encrypt

Connected!

OK



Sau đó Alice sẽ tạo p , g và một số bí mật a , tính $g^a \bmod p$ và gửi cho Bob (chỉ random 64 bit để chương trình chạy nhanh, đã từng thử với 128 bit và chạy mất 15 phút để tính primitive root):

The screenshot shows the 'Alice' interface with the following elements:

- Key is:** A text input field.
- Server:** 192.168.254.101
- Prime number p:** 9017719964451826729. A red box highlights the **Generate** button next to it.
- Primitive root g:** 7. A red box highlights the **Calculate** button next to it.
- Secret Number a:** 142857. A red box highlights the **Send $g^a \bmod p$** button.
- Annotations:**
 - Red text: "3 chọn một số bí mật a và gửi Modular exponentiation đi"
 - Red text: "1 Server đầu tiên tạo số nguyên tố p ngẫu nhiên"
 - Red text: "2 sau đó tính primitive root"
- Buttons:** Send encrypt, Send non-encrypt.

Alice tiếp tục gửi p và g cho Bob bằng chat không mã hóa:

The screenshot shows the 'Alice' interface with the following elements:

- Key is:** A text input field.
- Server:** 192.168.254.101
- Prime number p:** 9017719964451826729. A red box highlights the **Generate** button next to it.
- Primitive root g:** 7. A red box highlights the **Calculate** button next to it.
- Secret Number a:** 142857. A red box highlights the **Send $g^a \bmod p$** button.
- Chat Log:** A red oval highlights the chat messages: "Me: p is 9017719964451826729" and "Me: g is 7".
- Buttons:** Send encrypt, Send non-encrypt.



ở phía bên kia Bob nhận được các thông tin Alice gửi tiến hành nhập dữ liệu

The screenshot shows the Bob interface. At the top, it says "Bob". Below this, there are input fields for "Key is:", "Prime number p:", and "Primitive root g:". The "Key is:" field is empty. The "Prime number p:" field contains "9017719964451826729". The "Primitive root g:" field contains "7". To the right of these fields are buttons labeled "Accept p" and "Accept g", both of which are highlighted with red boxes. Above the "Accept p" button is a "Calculate key" button. To the right of the "Accept p" button is a "Secret Number b:" field containing "123456", with a "Send $g^b \text{ mod } p$ " button next to it. Below these fields are two large text areas: "Ciphertext:" and "Plaintext:". The "Ciphertext:" area contains the text: "ModularExponentiation received!", "Alice: p is 9017719964451826729", "Alice: g is 7", and "Me: ok". The "Plaintext:" area contains the text: "My IP address is 192.168.254.101", "192.168.254.101:20199 connected", "ModularExponentiation received!", "Alice: p is 9017719964451826729", "Alice: g is 7", and "Me: ok". At the bottom right, there are two buttons: "Send encrypt" and "Send non-encrypt".

Sau khi nhận được Modular exponentiation Alice tiến hành tính key, Bob cũng vậy:

The screenshot shows the Alice interface. At the top, it says "Alice". Below this, there are input fields for "Key is:", "Server", "Prime number p:", and "Primitive root g:". The "Key is:" field contains "3609979548214372776". The "Server" field contains "192.168.254.101". The "Prime number p:" field contains "9017719964451826729". The "Primitive root g:" field contains "7". To the right of these fields are buttons labeled "Calculate key", "Connect", "Generate", and "Calculate". The "Calculate key" button is highlighted with a red box. To the right of the "Calculate key" button is a "Secret Number a:" field containing "142857", with a "Send $g^a \text{ mod } p$ " button next to it. Below these fields are two large text areas: "Ciphertext:" and "Plaintext:". The "Ciphertext:" area contains the text: "Me: p is 9017719964451826729", "Me: g is 7", "Bob: ok", and "ModularExponentiation received!". The "Plaintext:" area contains the text: "Me: p is 9017719964451826729", "Me: g is 7", "Bob: ok", and "ModularExponentiation received!". At the bottom right, there are two buttons: "Send encrypt" and "Send non-encrypt".



Alice

Key is:

3609979548214372776

Calculate key

Server

192.168.254.101

Connect

Prime number p:

9017719964451826729

Generate

Primitive root g:

7

Calculate

Secret Number a:

142857

Send $g^a \text{ mod } p$

Me: p is 9017719964451826729

Me: g is 7

Bob: ok

ModularExponentiation received!

Me: hi

Me: we chat and encrypt message with our key now

Bob: [Y_!KV@

Bob: _\$VMi!rJiWQB YQACWOCiIDY^QC!NGiBjUZ YFROB

Me: p is 9017719964451826729

Me: g is 7

Bob: ok

ModularExponentiation received!

Me: hi

Me: we chat and encrypt message with our key now

Bob: hoo-ray

Bob: left is ciphertext, right is plaintext

Send encrypt

Send non-encrypt

Bob

Key is:

3609979548214372776

Calculate key

Prime number p:

9017719964451826729

Accept p

Primitive root g:

7

Accpet g

Secret Number b:

123456

Send $g^b \text{ mod } p$

Ciphertext:

ModularExponentiation received!

Alice: p is 9017719964451826729

Alice: g is 7

Me: ok

Alice: [

Alice: DS!iZQVM!iUVViQjT@NGBi[UJJV^PiQ[E\XGEi]VO!iWV@i

Me: hoo-ray

Me: left is ciphertext, right is plaintext

Plaintext:

My IP address is 192.168.254.101

192.168.254.101:20199 connected

ModularExponentiation received!

Alice: p is 9017719964451826729

Alice: g is 7

Me: ok

Alice: hi

Alice: we chat and encrypt message with our key now

Me: hoo-ray

Me: left is ciphertext, right is plaintext

Send encrypt

Send non-encrypt



Trong một lần trao đổi khác:

Bob

Key is:

Calculate key

Prime number p:

Accept p

Primitive root g:

Accept g

Secret Number b:

Send $g^b \bmod p$

Ciphertext:

Plaintext:

Alice: p is 4952857991538791749
Alice: g is 10
ModularExponentiation received!
Me: ok
Alice: ok
Alice: R^{XXY}
Me: ahihi
Alice: ZIJ_FQIKWA
Me: me too

My IP address is 192.168.254.101
192.168.254.101:18672 connected
Alice: p is 4952857991538791749
Alice: g is 10
ModularExponentiation received!
Me: ok
Alice: ok
Alice: ahihi
Me: ahihi
Alice: i love you
Me: me too

Send encrypt

Send non-encrypt

Alice

Key is:

Calculate key

Server:

Connect

Prime number p:

Generate

Primitive root g:

Calculate

Secret Number a:

Send $g^a \bmod p$

Me: p is 4952857991538791749
Me: g is 10
ModularExponentiation received!
Bob: ok
Me: ok
Me: ahihi
Bob: R^{XXY}
Me: i love you
Bob: ^SID_

Me: p is 4952857991538791749
Me: g is 10
ModularExponentiation received!
Bob: ok
Me: ok
Me: ahihi
Bob: ahihi
Me: i love you
Bob: me too

Send encrypt

Send non-encrypt

Trang 26 / 29

5. Task 3.4

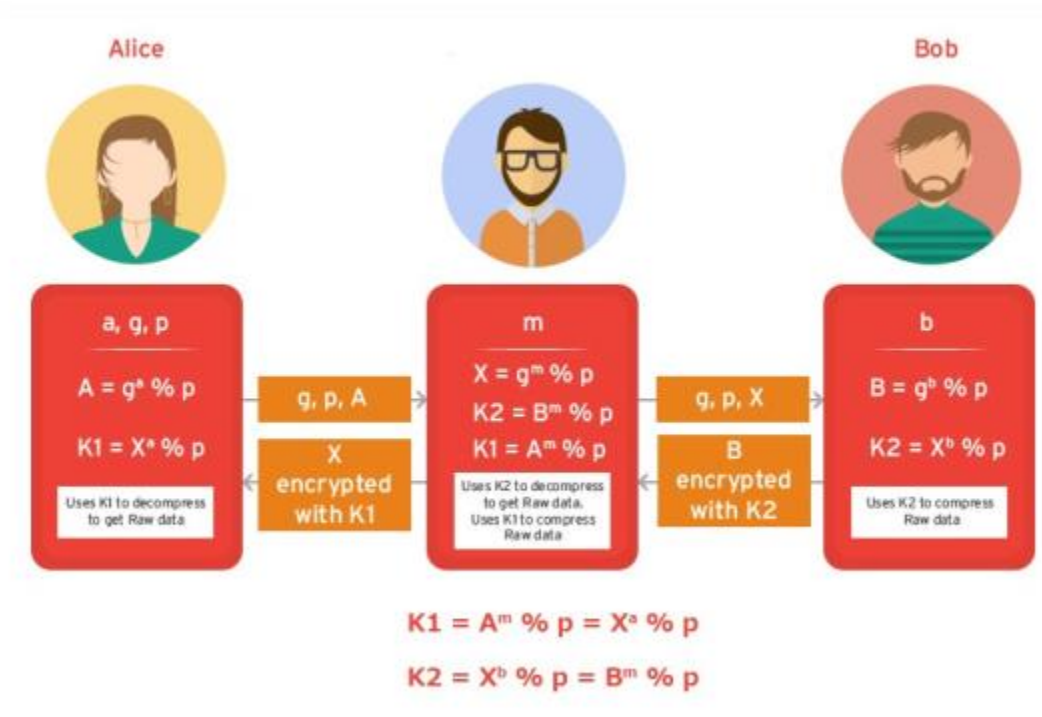
Trong giao thức trao đổi khóa Diffie-Hellman (DH), nếu kẻ tấn công có các giá trị g ; p ; $g^a \bmod p$; $g^b \bmod p$ thì khóa bí mật đang trao đổi có bị lộ không? Tại sao?

Khi có các giá trị g , p , $g^a \bmod p$, $g^b \bmod p$ thì khóa bí mật vẫn **rất khó bị lộ** bởi phép tính mod không khả nghịch, không thể suy ngược ra a , b . Thông thường giá trị của p khá lớn (khoảng 2048 bit) và thông thường sẽ dùng Sophie Germain prime q để tính $p = 2q + 1$ nên việc dò số gần như bất khả thi nên khóa bí mật khó bị lộ.

DH có hoàn toàn an toàn không? Mô tả hình thức tấn công và cách hạn chế (nếu có)

DH Không hoàn toàn an toàn, thuật toán Pohlig–Hellman có thể dùng để tìm a hoặc b từ các Modular exponentiation trao đổi, do đó số g thường chứa prime factor lớn để tránh bị thuật toán này bẻ khóa.

Ngoài ra có một phương thức tấn công phổ biến nhằm vào phương thức trao đổi khóa này là man-in-the-middle attack.





Một ví dụ về cách tấn công này:

1. Alice gửi tin cho Bob, tin này bị chặn bởi Mallory:
Alice "Hi Bob, it's Alice. Give me your key." → Mallory Bob
2. Mallory chuyển tiếp tin nhắn cho Bob; Bob không chắc rằng nó đến từ Alice:
Alice Mallory "Hi Bob, it's Alice. Give me your key." → Bob
3. Do đó Bob gửi public key cho Alice:
Alice Mallory ← [Bob's key] Bob
4. Mallory thay bằng key của chính mình và gửi cho Alice, nói rằng đây là Key của Bob:
Alice ← [Mallory's key] Mallory Bob
5. Alice mã hóa bằng key nhận được và tin rằng nó của Bob, nghĩ rằng chỉ Bob đọc được:
Alice "Meet me at the bus stop!" [encrypted with Mallory's key] → Mallory Bob
6. Tuy nhiên, key đó của Mallory và Mallory có thể giải mã nó, đọc và chỉnh sửa nó (nếu cô ta thích làm vậy), rồi lại mã hóa với key của Bob, và gửi cho Bob:
Alice Mallory "Meet me at the van down by the river!" [encrypted with Bob's key] → Bob
7. Bob nghĩ nó đến từ Alice.
8. Bob đi tới xe tải bên bờ sông và thay vì gặp Alice thì Bob gặp Mallory và bị bắt cóc.
9. Alice không biết Bob bị bắt cóc bởi Mallory chỉ nghĩ Bob tới trễ.
10. Sau khoảng thời gian lâu không thấy Alice phát hiện có chuyện gì đó xảy ra với Bob.

Kiểu tấn công này có điểm yếu là phải bằng một cách nào đó xen giữa vào cuộc trò chuyện của hai người mới thực hiện được, và đồng thời, nếu đang trong quá trình trò chuyện mà người ở giữa rời khỏi, hoặc sau một khoảng thời gian thì hai người kia sẽ biết được mình đã bị tấn công.

6. Tài liệu tham khảo

- How many bytes are required to hold N decimal digits:
<https://stackoverflow.com/questions/9443236/how-many-bytes-are-required-to-hold-n-decimal-digits>
- Why random? <https://crypto.stackexchange.com/questions/726/what-is-the-use-of-real-random-number-generators-in-cryptography>
- What type of cipher is RSA? <https://security.stackexchange.com/questions/1878/what-type-of-cipher-is-rsa>
- Man-in-the-middle: https://en.wikipedia.org/wiki/Man-in-the-middle_attack
- Diffie–Hellman key exchange:
https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange#Security
- XOR Encryption <https://www.programmingalgorithms.com/algorithm/xor-encryption>



- Find primitive root: <https://www.geeksforgeeks.org/primitive-root-of-a-prime-number-n-modulo-n/>
- Miller Rabin: <http://snipd.net/primality-testing-with-fermats-little-theorem-and-miller-rabin-in-c>
- Cryptool source: <https://www.cryptool.org/trac/CrypTool2/browser/trunk?order=name>