**2 MÃ CÔNG KHAI**

**MẬT MÃ KHÓA CÔNG KHAI**

**Nguyên lý thiết kế**

***Nguyên lý***

Như ta đã biết, mã khóa công khai, gọi tắt là mã công khai (*public key cryptosystem*) là hệ mã trong đó, mã hóa và giải mã sử dụng các khóa khác nhau. Khóa mã hóa (*encryption key*), ký hiệu **e**, được công bố công khai (*public key*) để ai cũng có thể sử dụng mã hóa thông điệp muốn gửi mật đến người có khóa cá nhân (*private key*), ký hiệu **d**, tương ứng với khóa công khai **e**. Cặp khóa (**e**, **d**), cặp số bất kỳ mà chúng có quan hệ toán học với nhau, ký hiệu R(e, d), sao cho biết **d** có thể suy ra được **e** nhưng ngược lại, biết **e** không thể suy ngược lại được **d**. Đây chính là nguyên lý để xây dựng một hệ khóa công khai.

**Nguyên lý**: hệ mã công khai được xây dựng đặt trên nền tảng một quan hệ toán học sao cho có thể công bố khóa công khai mà không thể suy ngược lại được khóa cá nhân.

***Hàm 1-chiều có cửa mật***

Hàm là 1-chiều có cửa mật (*one-way function with trapdoor*) nếu

. là dễ tính (độ phức tạp tính toán thấp), và

. là khó tính (độ phức tạp tính toán rất cao), nhưng

. sẽ dễ tính được nếu có thông tin cửa mật (trapdoor).

Hàm 1-chiều được *f* có tính chất như trên có thể được dùng để cài đặt quan hệ R(e, d), quan hệ giữa khóa công khai e và khóa cá nhân d của một hệ mã công khai, .

Phương pháp chính để xây dựng một hàm 1-chiều có cửa mật là

. Chọn hoặc xây dựng một bài toán khó (hard problem).

. Xây dựng quan hệ . Công bố e và giữ thông tin cửa mật tính được .

***Bài toán khó***

Bài toán khó (*hard problem*) là bài toán hay vấn đề có thuật toán được chứng minh là hầu như không thể thi hành thuật toán để có kết quả trong thời gian chấp nhận được.

Các bài toán sau được chấp nhận là khó và được sử dụng phổ biến trong mã hóa - mật mã.

*Bài toán phân tích ra thừa số* – **IFP**

Bài toán phân tích ra thừa số, đầy đủ là ‘bài toán phân tích số nguyên ra thừa số’ – **IFP** (*Integer Factorization Problem*) là bài toán phân tích hợp số n thành các thừa số nguyên tố.

Định lý cơ bản của số học phát biểu rằng mọi số nguyên lớn hơn 1 đều có thể viết dưới dạng tích hữu hạn các số nguyên tố (*prime number*) nhỏ hơn bằng nó. Cách viết này là duy nhất, không xét đến thứ tự các thừa số.

**. Số nguyên tố** (*prime*). Số nguyên được gọi là nguyên tố (prime) nếu . Trong đó gcd(a,b) là ước chung lớn nhất của a và b.

**. Ước số** (*divisor*). Số nguyên dương d được gọi là ước của số nguyên a nếu .

**. Ước chung** và **ước chung lớn nhất** (*common divisor* and *greatest common divisor*). Số nguyên dương d là ước chung của 2 số nguyên a và b nếu d là ước của a và d cũng là ước của b. Nếu d là ước chung lớn nhất trong các ước chung của a và b thì d được gọi là ước chung lớn nhất (*greatest common divisor*) của a, b, ký hiệu d = gcd(a, b).

Ký hiệu

**.** là tập tất cả các số nguyên tố nhỏ hơn *n*.

**.** là tập tất cả các số nguyên tố.

Thuật toán gcd(a, b) tính ước chung lớn nhất của 2 số nguyên dương sẽ được trình bày trong các phần sau.

**Định lý** (*định lý cơ bản của số học* – *Fundamental Theorem of arithmetic*).

sao cho

(\*)

Cách biểu diễn (\*) là duy nhất.

Từ các khái niệm trên, bài toán phân tích ra thừa số trong mã hóa-mật mã được phát biểu như sau:

**IFP** (*bài toán phân tích ra thừa số – Integer Factorization Problem*).

là 2 số nguyên tố lớn (*large prime*) có giá trị xấp xỉ nhau, và ,

. Cho , việc tính tích là dễ, nhưng

. Cho , việc tính lại hay là hầu như không thể.

Như vậy,

là hàm 1-chiều, và có được dùng làm cửa mật, ta có hàm 1-chiều có của mật.

Thực vậy, phép tính tích 2 số nguyên tố là dễ thực hiện trên máy tính với độ phức tạp của phép nhân 2 số lớn. Để tìm lại *n* khi không biết p và q, phải thực hiện phép chia thử. Nhưng nếu biết *q*, chỉ phải thực hiện 1 phép chia, *p = n/q*, tương đương với 1 phép nhân 2 số lớn.

*Bài toán log rời rạc* – **DLP**

Bài toán log rời rạc – **DLP** (*Discrete Logarithm Problem*) là bài toán tính giá trị của số mũ *x* khi được cho giá trị *y* của một phép tính lũy thừa theo cơ số *a* cho trước, *y = ax*.

Việc tính *x*, từ biểu thức *y* = *ax* là không khó trong tập các số thực, *x = logay*, vì thế bài toán được gọi là bài toán log (*logarithm problem*).

Nhưng nếu trong tập rời rạc lớn của các số nguyên, bài toán tính log rời rạc (discrete logarithm problem) được công nhận là bài toán khó.

Trong mã hóa-mật mã, bài toán DLP được phát biểu như sau:

**DLP** (*bài toán log rời rạc – Discrete Logarithm Problem*).

là số nguyên tố lớn, và *g* là số thỏa *gd* mod *p* ≠ 1, ∀*d* = *2, …, p*, và *gp* mod *p* = 1.

. Cho *g, p,* và *x* là số nguyên, việc tính *y = gx* mod *p* là dễ, nhưng

. Cho *g, p,* và *y* là số nguyên dương, việc tính *x* sao cho *gx* mod *p = y* là hầu như không thể.

Như vậy,

là hàm một chiều, và với *x* làm cửa mật, ta có hàm 1-chiều chiều với cửa mật.

Các bài toán như

. Bài toán tổng con – **SSP** (*sub-set problem*), mà tìm các số trong dãy nhiều số phân biệt cho trước sao cho tổng của chúng bằng một giá trị cho trước, cũng được công nhận là bài toán khó.

. Bài toán phân tích ma trận – **MFP** (*matrix factorization problem*), mà tìm ma-trận A sao cho AB = C khi chỉ được cho ma trận C, cũng có thể được sử dụng trong mã hóa-mật mã.

Tuy nhiên,

hai bài toán **IFP** và **DLP** là 2 bài toán được sử dụng phổ biến nhất trong mã hóa-mật mã.

**Phương pháp thiết kế hệ mã công khai**

Cuối cùng, để thiết kế một hệ mã công khai, quy trình tổng quát như sau:

**Lược đồ – GS** (*General Scheme*)

(**bước 1**) Chọn hay xây dựng một bài toán khó P.

(**bước 2**) Thiết kế một hàm 1-chiều với cửa mật *f*.

(**bước 3**) Thiết kế hàm mã và giải mã với cửa mật làm khóa cá nhân và các thông tin có thể công bố của hàm *f* là khóa công khai.

**HỆ MÃ CÔNG KHAI RSA**

**Định lý RSA**

Hệ mã công khai RSA, đặt theo tên của 3 nhà toán học **R**ivest, **S**hamir, và **A**dman, có thể được phát biểu qua định lý sau.

**Định lý RSA**. Cho p và q là 2 số nguyên tố phân biệt, p ≠ q. Đăt n = pq, ϕ = ϕ(n) = (p – 1)(q – 1). e và d là 2 số nguyên thỏa ed mod ϕ = 1. m và c là các số nguyên trong tập . Nếu c = me mod ϕ thì m = cd mod p.

Định lý RSA có thể dễ dàng chuyển thành giao thức để bảo mật thông tin.

**Giao thức RSA**

Trong giao thức này, giả sử Bob muốn chuyển cho Allice một tin mật của thông điệp rõ *m* dài *λ* bít. Ta có thể xem *m* như một số nguyên trong tập , với . Allice và Bob thi thành giao thức sau:

***Giao thức***

(**I**) Sinh khóa – **KeyGen(λ)**

Allice trước hết, chọn ngẫu nhiên 2 số nguyên tố λ/2 bit và tính:

(1) .

(2) .

(3) .

(4) Allice công bố khóa (e, n) và giữ bí mật d.

(**II**) Mã hóa – **Encryption(e,n,m)**

Để mã hóa thông điệp , Bob sử dụng khóa công khai e, tính và chuyển cho Allice kết quả tính toán

(5) .

(**III**) Giải mã – **Decryption(d,n,c)**

Để giải mã bản mã , Allice sử dụng khóa cá nhân của mình và tính

(6) .

Định lý RSA đảm bảo m tính được ở bước (6) chính là thông điệp rõ ban đầu của Bob.

***Độ bảo mật của giao thức***

Trong giao thức trên, tin tặc Cain có thể có được các thông tin công khai (e, n), và bản mã c khi trao đổi trên kênh không an toàn. Để tính được m, Cain có thể tìm trực tiếp m hay tìm d thì sẽ tìm được m.

*Tấn công từng phần* – **Partial attack**

Tính trực tiếp m từ c, e, n bằng cách giải bài toán:

,

mà là không thể vì không có d (bài toán cũng được xem khó như bài toán log rời rạc – DLP).

*Tấn công toán phần* – **Full attack**

Tìm khóa bí mật d từ e và n bằng cách giải bài toán

.

Muốn vậy phải xác định được

*phi* = (p – 1)(q – 1) = pq – p – 1 = n – p – q.

Để tính được *phi*, phải giải IFP, là không thể.

***Cài đặt giao thức RSA***

Giao thức RSA làm việc trên các số lớn trong tập . Về mặt toán học, với các phép toán +n và \*n (modulo n) trên các số nguyên tạo thành cấu trúc vành (*ring*), vành .

*Tính toán số lớn* – **BigInteger**

. **addMod(a, b, c, n)**: phép cộng (mod n), ký hiệu

.

Hàm addMod có thể được cài đặt dễ dàng bằng cách cộng có nhớ trong hệ nhị phân.

. **mulMod(a, b, c, n)**: phép nhân (mod n), ký hiệu

.

Hầu hết ngôn ngữ lập trình không hỗ trợ phép nhân trên các số lớn (hàng ngàn bit), nên phép modulo (%) chỉ là ký hiệu, và ta cần một thuật giải hiệu quả để tính kết quả ab%n này.

Có nhiều thuật toán cho phép nhân số lớn này. Ở đây ta sẽ tính trên chuỗi bit và cài đặt phương pháp tính theo những ghi chú như sau.

. x, y, n là những số lớn với 0 ≤ x, y < n. là biểu diễn nhị phân b bít, thì giá trị thập phân của y là .

. .

. .

Ta mã giả thuật giải mulMod như sau

|  |
| --- |
| mulMod(x, y, n):  P = 0.  if y0 = 1: P = x.  for i = 1 → b – 1:  x ≡ 2\*x (mod n)  z = yi\*x  P ≡ (P + z) (mod n)  return P |

. **powerMod(x, p, n)**: phép lũy thừa (mod)

Cho và số nguyên b bit có biểu diễn nhị phân . Ta có

. .

. .

Và ta có mã giả sau

|  |
| --- |
| powerMod(x,a,n):  y = 1  if a == 0:  return y  A = x  if a0 == 1: y = x  for i = 1 → b – 1:  A ≡ A2 (mod n)  if ai == 1: y ≡ y\*A (mod n)  return y |

Ta cũng có thể tính từ bit cao xuống bit thấp như mã giả sau

|  |
| --- |
| powerMod(x,a,n):  y = 1  for i = b – 1 ← 0:  y ≡ y2 (mod n)  if ai == 1: y ≡ y\*x (mod n)  return y |

*Thuật giải Euclide* – **Euclide algorithm**

. **keyGen(e, d, phi)**: sinh khóa e và d thỏa ed mod phi = 1.

Đẳng thức

. (\*)

. Hai số thỏa được gọi là các số khả nghịch (mod k) – invertible (mod k), và *y* là nghịch đảo (mod k) của *x*, ký hiệu . Tương tự,

. Tập là tập tất cả các phần tử khả nghịch (mod k).

Như vậy, để tạo khóa (e, d) ta cần tìm phần tử e khả nghịch (mod ϕ) và tính d ≡ e-1 mod ϕ. Ta có kết quả sau:

Mệnh đề. Số nguyên , nghĩa là x nguyên tố cùng nhau với k.

Ta đã biết cách tính ước chung lớn nhất của 2 số nguyên dương sử dụng kết quả của Euclide, như mã giả sau:

|  |
| --- |
| gcd(a, b):  while a ≠ b:  if a > b: a = a – b  else: b = b – a  return a |

Tuy nhiên, ta cần một thuật toán hiệu quả để tính ước chung lớn nhất của 2 số nguyên dương lớn dài b bit. Ta cài đặt định lý sau.

**Định lý** (*thuật giải Euclide nhị phân*).

Cho là biểu diễn dạng tích các số nguyên tố theo định lý cơ bản của số học, của 2 số nguyên dương *x, y* ≥ 2.

Đặt , ta có

. .

. .

. .

. .

Ta mã giả định lý trên như sau

|  |
| --- |
| gcd(x, y):  g = 1.  while (cả x và y đều chẵn):  x = x/2.  y = y/2.  g = 2\*g.  while x > 0:  while (x chẵn): x = x/2.  while (y chẵn): y = y/2.  t = |x – y|.  if (x > y): x = t.  else: y = t.  g = g\*y.  return g. |

Bezout, nhà toán học Pháp, sau này đã mở rộng định lý Euclide bằng định lý mang tên ông.

**Định lý** (*Bezout*). Mọi số nguyên dương a, b, luôn có thể biểu diễn dưới dạng

.

Như vậy, có thể sử dụng định lý Bezout để tạo cặp khóa (e, d) cho giao thức RSA, như mã giả sau:

|  |
| --- |
| keyGen(phi):  while 1 > 0:  sao cho *e* là số lẻ.  if Bezout(e, d, phi, y) == 1: #(\*)  return e, d |

Thực vậy, nếu điều kiện (\*) thỏa, ta có

.

Theo định nghĩa, .

*Số nguyên tố* – **Prime number**

Có nhiều thuật toán tạo ngẫu nhiên một số nguyên tố lớn b bit. Ở đây, ta sử dụng ý tưởng cơ bản sau:

**primeGen(b)**.

Phát sinh một số lẻ p có b bít và kiểm tra xem p có phải là số nguyên tố hay không. Nếu không, quay lại tạo số lẻ khác. Quá trình cứ được lặp đi lặp lại cho đến khi được số nguyên tố.

Thuật toán trên phụ thuộc chính vào việc kiểm tra xem một số lẻ có phải là số nguyên tố hay không. Như ta đã biết, dùng phép chia thử, cần cỡ √p phép chia thử. Ta sẽ dùng định lý Fermat nhỏ để xây dựng thuật giải kiểm tra một số có phải nguyên tố hay không.

**Định lý** (*Fermat nhỏ*). Số nguyên dương p là nguyên tố nếu và chỉ nếu với mọi cơ sở là số nguyên b, b không là bội của p, ta có .

Ta có một số nhận xét sau:

* Mọi số nguyên tố p lớn hơn 2 đều là số lẻ: p = 2m + 1, với 1 m nào đó.
* p = 2m + 1 là số nguyên tố nếu chỉ nếu với mọi 1 < b < p, b2m + 1 ≡ 1 (mod p) hay bm = ±1 (mod p).
* Nếu p = 2m + 1 và có một 1 < b < p sao cho bm ≠ ±1 (mod p) thì p không phải số nguyên tố.

Từ những nhận xét trên, ta nói một số là giả-nguyên tố cơ sở b nếu nó thỏa định lý Fermat nhỏ. Cụ thể

. Cho p – 1 = 2rm, với m là số lẻ, và b là một cơ sở (base) nguyên bất kỳ. Nếu có một k sao cho thì p không nguyên tố; ngược lại, ta gọi p là giả nguyên tố (pseudo-prime) cơ sở b.

. Một số p được gọi là giả nguyên tố mạnh (*strong pseudo-prime*) nếu nó giả nguyên tố của nhiều cở sở b khác nhau.

. Số p cũng được gọi là giả nguyên tố mạnh nếu nó giả nguyên tố của một cơ sở ngẫu nhiên bất kỳ.

Thuật giải sau kiểm tra một số giả nguyên tố mạnh.

|  |
| --- |
| primeTest(p):  .  Tìm r và số m lẻ sao cho p – 1 = 2rm.  for k = 1 → r:  if return FALSE.  return TRUE. |

**Chứng minh định lý RSA**

Trươc khi chứng minh định lý RSA, ta chứng minh định lý số dư Trung hoa như sau.

**Định lý CRT** (*Chinese Remainder Theorem*). là k số nguyên dương đôi một nguyên tố cùng nhau, , và là k số nguyên. Hệ (\*) sau có duy nhất nghiệm trong :

.

***Chứng minh về sự tồn tại****.*

(1) Đặt

.

#Do , nên ni khả nghịch (mod m), i = 1, 2, …, k.

(2) Đặt .

#Ta có .

(3) Trả về *x*.

***Chứng minh RSA***

Ta cần chứng minh .

Ta có,

* Nếu , ta có

(sử dụng định lý Fermat nhỏ).

Rõ ràng là nghiệm của hệ trên. Và theo định lý số dư Trung Hoa, hệ có nghiệm duy nhất, nên m là duy nhất.

* Nếu m = p thì med ≡ ped ≡ 0 (mod p) và med ≡ ped ≡ p (mod q) (định lý Fermat).

Vậy, trong mọi trường hợp cd ≡ med ≡ m (mod n).

**Giải mã nhanh RSA**

Do người giải mã là người sinh cặp khóa (e, d), nên đó là người biết mọi thành phần tạo nên (e, d), nghĩa là biết p và q. Sử dụng tri thức về p và q, ta có thể tính nhanh cd (mod n) như sau.

Xét phép tính cd (mod n), với n = pq là tích của 2 số nguyên tố phân biệt p và q.

Đặt d1 = d mod (p – 1) và d2 = d mod (q – 1).

Có các số k1 và k2 sao cho d = d1 + k1(p – 1) = d2 + k2(q – 1).

Thì,

Theo CRT, hệ trên có duy nhất nghiệm trong n = pq.

Thuật giải giả mã nhanh RSA

|  |
| --- |
| fastDecryption(c,p,q):  d1 = d mod (p – 1)  d2 = d mod (q – 1)  a1 = powerMod(c, d1, p)  a2 = powerMod(c, d2, q)  m = CRT(a1, a2, p, q) |

**HỆ MÃ CÔNG KHAI DIFFIE-HELLMAN**

**Giao thức thiết lập khóa**

Khác với hệ mã RSA mà bảo mật dựa trên độ khó bài toán phân tích ra thừa số, Diffie-Hellman, đặt theo tên của 2 nhà toán học phát minh ra, là hệ mã mà bảo mật dựa trên bài toán log-rời rạc – **DLP** (*Discrete Logarithm Problem*).

Thực ra, Diffie-Hellman được thế kế cho mục đích thiết lập bí mật chung giữa 2-đối tác, được gọi là giao thức trao đổi (thực ra là thiết lập) khóa (*key exchange protocol*). Trong giao thức này, Allice và Bob, mỗi người chia sẻ bí mật riêng để thiết lập khóa bia mật chung của cả hai.

|  |  |
| --- | --- |
| **Key-Exchange Protocol** | |
| **Allice** | **Bob** |
| (**I**) **Tạo thông số dùng chung**  # Allice và Bob sử dụng chung trường , với p là số nguyên tố lớn b bit, và phân tử sinh g.  (1) .  (2) #chọn phần tử sinh g | |
| (**II**) **Trao đổi các thông tin** |  |
| (3) # Allice chọn số bí mật x cho mình  (4)  (5) gửi kA cho Bob | (3’) #Bob cũng chọn cho minh số y  (4’)  (5’) gửi kB cho Allice |
| → kA | kB ← |
| (**III**) **Thiết lập khóa bí mật chung** |  |
| (6) | (6’) |
| Ta có k = k’, thực vậy  . | |

**Cơ sở toán học của giao thức Difie-Hellman**

***Cơ sở toán học***

Độ an toàn của giao thức Diffie-Hellman dựa trên giả thiết cùng tên, giả thiết Difie-Hellman (**DDH** – *Decision* *Diffie-Hellman Assumption*).

Xét một nhóm tuần hoàn (nhân) G có bậc nguyên tố p và với phần tử sinh g. Giả thiết **DDH** phát biểu rằng, với ga và gb với được chọn ngẫu nhiên và độc lập, giá trị gab "*cũng như*" một phần tử ngẫu nhiên trong G.

***Cấu trúc nhóm***

Một tập hợp G khác rỗng, được trang bị phép toán nhân, \*, được định nghĩa bởi

, có các tính chất

(**.**) Tính đóng (*closed*): .

(**.**) Phần tử đơn vị (*Identity*): .

(**.**) Tính khả nghịch (*Inverse*): .

(**.**) Tính kết hợp (*Associativity*): .

Thì (G, \*) được gọi là nhóm (group).

***Ví dụ***

**Ví dụ 1**. Tập các số nguyên với phép cộng thông thường là nhóm, . Thực vậy,

(.) .

(.) Phần tử đơn vị là 0: *x* + 0 = 0 + *x* = *x*, với mọi số nguyên *x*.

(.) –*x* là phần tử nghịch đảo của *x*.

(.) *x* + (*y* + *z*) = (*x* + *y*) + *z* = *z* + *y* + *z*.

**Ví dụ 2**. Tập , với p là số nguyên tố, với phép nhân modulo, \* (mod), là nhóm (. Thực vậy,

(.) .

(.) Phần tử đơn vị là 1: .

(.) và p là số nguyên tố, nên gcd(*x*, p) = 1, hay *x* khả nghịch.

(.) .

***Nhóm tuần hoàn***

*Nhóm con –* **sub-group**

Cho nhóm (G, \*) và H là tập con khác rỗng của G. Nếu (H, \*) là nhóm thì (H, \*) được gọi làn nhóm con (*sub-group*) của G.

Chẳng hạn, nhómvới là tập tất cả các số nguyên chẵn- là nhóm cón của nhóm (). Hay nhóm (H, \*) chỉ có phần tử đơn vị của nhóm (G, \*), H = {1}, là nhóm con của (G, \*). (G, \*) là nhóm con của chính nó.

Định lý Lagrange cho biết quan hệ giữa số phần tử của nhóm H con với nhóm G.

**Định lý** (*Lagrange*). H là nhóm con của G thì số phần tử của nhóm con H, gọi là bậc của H, ký hiệu O(H), là ước của bậc của G, O(H) | O(G).

Như vậy, nếu O(G) là một số nguyên tố thì G chỉ có 2 con, nhóm có đúng phần tử đơn vị, và chính nó.

*Phép lũy thừa* – **power**

Cho một phần tử *x* thuộc nhóm nhân (G, \*) và một số nguyên k, thực hiện tính kết hợp k lần *x* lại với nhau, ký hiệu là *x*k, được lũy thừa k của x.

Nếu k = 0, ký hiệu *x*0 = 1 là phần tử đơn vị của nhóm (G, \*).

Chẳng hạn, , thì . Hay , thi .

*Nhóm tuần hoàn* – **cyclic group**

Cho phần tử *x* thuộc nhóm nhân (G, \*),

. Nhóm con sinh bởi *x*, ký hiệu là nhóm con của G với .

. Nếu tập G hữu hạn, thì gọi là nhóm đơn sinh hay nhóm tuần hoàn (*cyclic group*) do sẽ có 1 số nguyên dương k sao cho *x*k = 1, hay < *x* > = {*x*0 = 1, *x*1 = *x*, *x*2 = *x*\**x*, …, *x*k-1, (*x*k = 1)}. k được gọi là bậc của phân tử *x,* ký hiệu o(*x*) = k.

. Nếu < g > = G, thì g được gọi là phần tử sinh (*generator*) của nhóm tuần hoàn G.

Như vậy, nếu < g > = G hữu hạn,

. {gi, i = 0, 1, …, k = |G|} là một hoán vị của các phần tử trong G.

. Nếu G có bậc nguyên tố thì G có phần tử sinh.

**Cài đặt giao thức Diffie-Hellman**

Ngoài thủ tục tìm phần tử sinh của nhóm , với p là số nguyên tố, ta cần hàm tạo ra phần tử sinh cho giao thức DH.

Do p là số nguyên tố, ta biết là nhóm nhân tuần hoàn, nên tồn tại phần tử sinh. Tuy nhiên, không phải phần tử nào thuộc cũng là phần tử sinh, chỉ những phần tử *x* có bậc bằng bậc của , o(*x*) = O = p – 1, mới là phần tử sinh.

Ví dụ, xét ,

. Với *x* = 2, ta có < 2 > = {20 = 1, 21 = 2 , 22 = 4 (, 23 = 1)} , nên 2 không phải phần tử sinh.

. Với *x* = 3, < 3 > = {30 = 1, 31= 3, 32 = 2, 33 = 6, 34 = 4, 35 = 5 (, 36 = 1)} . 3 là phần tử sinh.

Giả sử p nguyên tố và , thuật toán sau tìm phần tử sinh g của .

|  |
| --- |
| generator(p):  P ← {p1, …, pn}  while TRUE:    for i = 1 → n:  z = n/pi  if xz mod p == 1: break  return x |

**Phân tích hệ mã Diffie-Hellman**

***Log-rời rạc***

Như ta đã biết, sự bảo mật của hệ mã Diffie-Hellman dựa vào độ khó của bài toán log-rời rạc. Có nhiều phương pháp giải bài toán log-rời rạc đã được công bố, nhưng phương pháp Index-Calculus có thể được xem là phương pháp mạnh nhất giải bài toán log-rời rạc.

Trước khi mô tả thuật toán Index-Calculus, ta nhắc lại ở đây một số tính chất của log-rời rạc, mà trên nguyên tắc, thừa hưởng của logoarithm ta đã biết trên trường số thực.

**Mệnh đề**. Gọi p là số nguyên tố và g là phần tử sinh (mod p). Giả sử a và b là các số nguyên dương không chia hết cho p. Thì

(*i*) logg1 ≡ 0 (mod p – 1).

(*ii*) logg(ab) ≡ logga + loggb (mod p – 1).

(*iii*) loggak ≡ klogga (mod p – 1).

Ví dụ, p = 13 và phần tử sinh g = 2 (mod 13).

. log21 ≡ log2212 ≡ 12 ≡ 0 (mod 12).

. log2(9\*12 (mod 13)) ≡ log2 4 ≡ 2 (mod 12) = log29 + log212 ≡ 6 + 8 ≡ 2 (mod 12).

. log2(1210 (mod 13))= log2 (61917364224 (mod 13)) = log2 1 ≡ 12 (mod 12) = 10\*log212.

***Thuật toán index-calculus***

Thuật toán Index-Calculus tính tìm x = loggb (mod p) với gx ≡ b (mod p), khi được cho p, g và b. Thuật toán được mô tả gốm các bước sau.

(**Bước 1**) Chon từ dãy 2, 3, …, p – 2, một dãy con t số nguyên tố S = {p1, p2, …, pt} sao cho có nhiều phần tử gi, 1 ≤ i < p – 2, được biểu diễn như tích các phần tử trong S.

(**Bước 2**) Chọn ngẫu nhiên số j, 0 ≤ j < p – 2 và tính gj (mod p).

(Bước 3) Biểu diễn gj như tích các phần tử trong S:

.

(Bước 4) Lấy log (mod p) 2 vế biểu thức trên:

.

(**Bước 5**) Lặp lại các bước từ (Bước 2) đến (Bước 4) để tạo đủ hệ t phương trình, và giải hệ phương trình tuyến tính đó. Nếu hệ phụ thuộc tuyến tính, quay lại (Bước 2) để xây dựng lại hệ phương trình tuyến tính.

(**Bước 6**) Chọn ngẫu nhiên số k, 0 ≤ k ≤ p – 2, và tính gkb (mod p – 1).

(**Bước 7**) Biểu diễn gkb thành tích các phần tử trong S:

.

Nếu không biểu diễn được như trên thì quay lại (Bước 6).

(**Bước 8**) Lấy logg (mod p – 1) 2 vế phương trình trên:

(mod p – 1).

Suy ra

.

Ví dụ giải 6x ≡ 57 (mod 107).

- Chọn S = {2, 3, 5, 7}.

- Giả sử phát sinh số nguyên ngẫu nhiên, sau nhiều lần ta viết được

. 624 ≡ 42 (mod 107) → 42 = 2\*3\*7.

. 66 ≡ 4 (mod 107) → 4 = 22.

. 633 ≡ 15 (mod 107) → 15 = 3\*5.

. 634 ≡ 90 (mod 107) → 90 = 2\*32\*5.

- Lấy log6 2 vế, ta có

. 24 ≡ log62 + log6 3 + log67 (mod 106).

. 6 ≡ 2\*log62 (mod 106).

.33≡ log63 + log65 (mod 106).

. 34 ≡ log62 + 2\*log63 + log65 (mod 106).

- Giải hệ 4 phương trình trên ta được:

. log62 ≡ 3

. log63 ≡104

. log65 ≡ 35

. log67 ≡ 23

- Bây giờ, tính log657 (mod 107). Sau nhiều lần lấy ngẫu nhiên đươc k = 38 có thể biểu diễn được dưới dạng

57\*638 ≡ 35 (mod 107) → 35 = 5\*7.

- Lấy log6 (mod 106) 2 vế phương trình trên,

. log657 + 38 ≡ log65 + log67

≡ 35 + 23

≡ 58 (mod 106).

Vậy x = log657 = 58 – 38 = 20.

***Tấn công qua trung gian***

Theo giả thuyết Diffie-Hellman (DH conjecture) thì hầu như không thể tính được gxy chỉ từ gx, gy. Mặc dầu vậy, nếu số nguyên tố p là số Mersenne, p = 2q + 1, với q cũng là số nguyên tố, hệ DH có thể sẽ bị phá vỡ. Thực vậy, sử dụng kết quả của định lý sau:

**Định lý**. x là phần tử sinh của là số nguyên tố. Với φ(m) là hàm φ-Euler trả về số các số nguyên dương nhỏ hon m và nguyên tố cùng nhau vói m.

Nếu

p = 2q + 1, thì theo định nghĩa của hàm φ-Euler, φ(p) = 2q, và với phần tử sinh x ∈ (1, p) ta có x2 ≡ 1 (mod p) và xq ≡ 1 (mod p).

Vậy 1 = gp-1 ≡ (gq)2 (mod p) = β2.

Theo định lý trên, β ≠ 1 và β2 = 1, suy ra β = -1.

Khi đó, k ≡ gxyq  ≡ βxy (mod p) ∈ {-1, 1}.

Vì vậy, tin tặc có thể sử dụng tấn công trung gian (**MiM** attack – *Man in the Middle attack*), thay 2 thông điệp gx, gy bằng các thông điệp mới gxq, và gyq, thì khóa bí mật chung chỉ có 2 giá trị là -1 và 1.

**Hệ mã ElGamal**

Trong mô hình này, giả sử Bob muốn gửi thông điệp mật của thông điệp rõ m cho Allice. Bob và Allice thi hành giao thức sau trên trường , với p là số nguyên tố lớn:

|  |  |
| --- | --- |
| Allice | Bob |
| **Bước 1**: sinh khóa – **Key generating**  . p ← primeGen(λ) #tạo số nguyên tố p  . g ← generator(p) #chọn phần tử sinh g  . #chọn ngẫu nhiên 1 khóa cá nhân d  . e ← gd % p #tính và công bố khóa e | Cặp khóa (e, d) với (e, p, g) được công bố và d giữ bí mật. |
| x ngẫu nhiên làm cho hệ mã có tính an toàn ngữ nghĩa, nghĩa là cùng bản rõ, các lần mã khác nhau sẽ cho bản mã khác nhau. | **Bước 2**: mã hóa – **Encryption**  . #chọn ngẫu nhiên số nguyên x  . c1 ← gx (mod p)  . c2 ← m\*ex (mod p)  . C ← (c1, c2) #tính C và gửi C cho Allice |
| **Bước 3**: giải mã – **Decryption**  .  . *v* ← *u*-1 (mod p)  . m ← c2\*v (mod p) | Ta thấy |

**Độ an toàn của ElGamal**

Độ an toàn của hệ mã dựa trên giả thuyết Diffie-Hellman: DLP là bài toán khó.

Tuy nhiên, trong trường hợp x bị sử dụng lại, ElGamal có thể bị phá vỡ. Thực vậy, giả sử . Nếu biết m, c1 và c2, ta có thể suy ra m’ như sau:

*.*

Như vậy, để an toàn, hàm sinh số ngẫu nhiên phải thực sự ngẫu nhiên.

**MÃ ĐỒNG CẤU**

**Khái niệm đồng cấu**

Một hàm *f* xác định trên D được gọi là đồng cấu (*homomorpic*) theo phép toán ⊕ nếu

.

Khái niệm mã đồng cấu (*homomorphic encryption*) cũng được định nghĩa tương tự.

Hệ mã được gọi là mã đồng cấu theo phép toán nếu .

Mã đồng cấu được áp dụng cho những ứng dụng ở đó cần phải thực hiện một số tính toán trên bản rõ mà không phải giải mã các bản mã nhận được. Chẳng hạn các hệ hợp tác tính toán đảm bảo tính bí mật thông tin, hay các hệ xác minh mà không phải cung cấp thông tin bản rõ có thể sử dụng mã đồng cấu để hiện thực.

**Các hệ mã đồng cấu phổ biến**

***Hệ mã đồng cấu RSA***

RSA là hệ mã đồng cấu theo phép nhân modulo.

Thực vậy, giả sử c1 = RSAe,n(m1) và c2 = RSAe,n(m2)lần lượt là bản mã RSA của các thông điệp rõ m1 và m2 với khóa công khai e, n. Ta có

, thì

,

Hay  chính là bản mã của thông điệp .

***Hệ mã đồng cấu ElGamal***

ElGamal là hệ đồng cấu theo phép nhân modulo.

Thực vậy, giả sử C1 = ElGamale,g,p(m1) và C2 = ElGamale,g,p(m2)lần lượt là bản mã ElGamal của các thông điệp rõ m1 và m2 với khóa công khai e, g, p. Ta có

.

Đặt

.

Thì C chính là bản mã ElGamal của m = m1\*m2.

Một cách tổng quát,

ElGamal-like là mã đồng cấu.

***Hệ mã đồng cấu Paillier***

Bên cạnh 2 hệ mã công khai RSA và ElGamal-like, ta học them hệ mã công khai, cũng có tính đồng cấu, cũng được sử dụng phổ biến trong bảo mật thông tin, mã Paillier (*Paillier encryption*).

Hệ mã Paillier, được phát minh và đặt theo tên của Pascal Paillier vào năm 1999, là một thuật toán mã bất đối xứng xác suất mà độ bảo mật dựa trên bài toán tính lượng thứ n được cho là bài toán khó. Giả định về độ dư tổng hợp quyết định là giả thuyết về tính khó sửa mà hệ thống mật mã này dựa vào.

Lược đồ mã hóa Paillier được mô tả như sau.

*Sinh khóa* – **KeyGen**

(1) Chọn ngẫu nhiên 2 số nguyên tố lớn phân biệt p và q thỏa

gcd(pq, (p – 1)(q – 1)) = 1.

(2) Tính n = pq và λ = lcm(p – 1, q – 1), với lcm (*least common multiple*) là hàm tính bội số chung nhỏ nhất của 2 số nguyên dương.

(3) Chọn ngẫu nhiên số , với là tập các số khả nghịch (mod n2).

(4) Kiểm tra đẳng thức với L(x) ký hiệu số nguyên không âm lớn nhất sao cho (x – 1) ≥ vn.

Khóa công khai sẽ là e = (n, g), và kháo cá nhân là (λ, μ).

Lưu ý: nếu p và q có chiều dài bit như nhau, đơn giản chỉ cần chọn g = n + 1, λ = φ(n) = (p – 1)(q – 1), và μ ≡ φ(n)-1 (mod p).

*Mã hóa* – **Encryption**

Để mã hóa thông điệp m, 0 ≤ m < n.

(1) Chọn ngâu nhiên số nguyên r, 0 < r < n nguyên tố cùng nhau với n, gcd(n, r) = 1.

(2) Tính .

c chính là bản mã của m: c = Pailliern,g(m).

*Giải mã* – **Decryption**

Để giải mã bản mã .

(1) Tính .

Thì m chính là thông điệp rõ của bản mã c.

*Các tính chất đồng cấu của Paillier* – **homomorphic properties**

Mã Paillier là mã đồng cấu với nhiều tính đồng cấu.

Thực vậy, ký hiệu D là hàm giải mã và E là hàm mã hóa.

Đồng cấu theo phép cộng

* Tích của 2 bản mã được giải mã thành tổng của 2 thông điệp.

D(E(m1, r1) \* E(m2, r2)) mod n2) = m1 + m2 mod n.

* Tích của bản mã của thông điệp này với lũy thừa theo g của thông điệp khác được giải mã thành tổng của 2 thông điệp.

D(E(m1, r1) \* gm2 mod n2) = m1 + m2 mod n.

Đồng cấu theo phép nhân

* Lũy thừa bản mã của thông điệp này với lũy thừa theo g của thông điệp khác được giải mã thành tích của 2 thông điệp.

D(E(m1, r1)m2 mod n2) = m1 \* m2 mod n,

D(E(m2, r2)m1 mod n2) = m1 \* m2 mod n.

Tổng quát hơn, một bản mã được nâng lũy thừa k sẽ giải mã thành tích của thông điệp và hằng số,

D(E(m1, r)k mod n2) = k \* m mod n.

Tuy nhiên, với mã hóa Paillier của hai thông điệp, không có cách nào để tính bản mã của tích 2 thông điệp này mà không biết khóa cá nhân.