Mật mã Khóa Công khai
Public Key Cryptosystems

Văn Nguyễn

Đại học Bách Khoa Hà nội

9/13/2008

Điểm yếu của hệ mã đối xứng

- Vấn đề quản lý khoá (tạo, lưu mật, trao chuyển ...) là nan giải trong môi trường trao đổi tin giữa rất nhiều người dùng.
- Không thể thiết lập được chữ ký điện tử
  - Do đó không thể đảm bảo non-repudiation[1] (không thể phủ nhận được) cho các giao dịch thương mại điện tử.
    - Dịch vụ non-repudiation: cung cấp bằng chứng để chứng gian những trường hợp phía bên kia chối bỏ một giao dịch nào đó, E.g. A chối đã không tiến hành giao dịch với B, mà giao dịch bị người khác mạo nhận A làm trái phép
    - Vì đối xứng, cần bên thứ ba có đủ uy tín làm trọng tài giao dich ➔ dễ bị quá tải

Ý tưởng của Diffie & Hellman

- Diffie & Hellman (1975-76) đã đề xuất một loại hệ mã với nguyên tắc mới, được gắn với một NSD nhất định chứ không phải là gắn với một cuộc truyền tin giữa một cặp NSD.
  - mỗi user có hai khoá: một khoá bí mật (secret key/private key) và một khoá công khai (public key) -- tự do phổ biến công khai.
  - Khoá thứ nhất gắn liền với giải mã, còn khoá thứ hai với sinh mã.
  - Hoạt động của chúng là đối xứng

$$X = D(z, E(Z, X)) \quad \text{hay } X = D_z E_Z (X) \qquad (1)$$

và $$X = E(Z, D(z, X)) \quad \text{hay } X = D_Z E_z (X) \qquad (2)$$

  - Trong đó (1) được sử dụng cho truyền tin mật: Còn (2) sẽ được sử dụng để xây dựng các hệ chữ ký điện tử (Ký bằng E_z và kiểm định bằng D_Z).
  - Hệ mã theo nguyên tắc nói trên được gọi là hệ mã với khoá công khai (public key cryptosystems - PKC) hay còn được gọi là mã phi đối xứng (asymmetric key cryptosystems).

Nguyên tắc cấu tạo một hệ PK (trapdoor)

- Một hệ mã PKC có thể được tạo dựng trên cơ sở sử dụng một hàm kiểu one - way (1 chiều). Một hàm f được gọi là one-way nếu:
  1. Đối với mọi X tính ra Y = f(X) là dễ dàng.
  2. Khi biết Y rất khó để tính ra X.
- Ví dụ. Cho n số nguyên tố p1, p2, ...pn ta có thể dễ dàng tính được N = p1 * p2 * ... * pn, tuy nhiên khi biết N, việc tìm các thừa số nguyên tố của nó là khó khăn hơn rất nhiều
- Cần một hàm one-way đặc biệt, trang bị một trap-door (cửa bẫy), sao cho nếu biết trap-door này thì việc tính X khi biết f(X) (tức là đi tìm nghịch đảo của f) là dễ, còn ngược lại thì khó
- Một hàm one-way có trap door như thế ➔ một hệ mã PKC
  - Lấy E_z (hàm sinh mã) là hàm one- way có trap-door
  - Trap- door chính là khoá mật, mà nếu biết nó thì có thể dễ dàng tính được cái nghịch đảo của E_Z tức là biết D_z, còn nếu không biết thì rất khó tính được.

Trapdoor Knapsack dựa trên bài toán đóng thùng

- 1978, hai ông Merkle - Hellman đã đề xuất một thuật toán mã hoá PKC dựa trên bài toán ĐÓNG THÙNG như sau:
  - Cho 1 tập hợp các số dương a_i, $1 \le i \le n$ và 1 số T dương. Hãy tìm 1 tập hợp chỉ số $S \subset \{1,2,...,n\}$ sao cho: $\sum_{i \in S} a_i = T$

- Bài toán này là một bài toán khó, theo nghĩa là chưa tìm được thuật toán nào tốt hơn là thuật toán thử-vét cạn
  - Thời gian xử lý vét cạn có thể tỉ lệ luỹ thừa theo kích thước input *n*.

  - VD: $(a_1, a_2, a_3, a_4) = (2, 3, 5, 7)$ $T = 7$.
    Như vậy ta có 2 đáp số $S = (1, 3)$ và $S = (4)$.

Hệ PKC Merkle - Hellman

- Từ bài toán đóng thùng này chúng ta sẽ khảo sát các khả năng vận dụng để tạo ra thuật toán mã khối PKC. Sơ đồ đầu tiên như sau:
  - Chọn một vector a = $(a_1, a_2, \ldots, a_n)$ - được gọi là vector mang (cargo vector)
  - Với một khối tin X = $(X_1, X_2, X_3 \ldots, X_n)$, ta thực hiện phép mã hoá như sau: $T = \sum a_i X_i$ (*)
  - Việc giải mã là: Cho mã T, vector mang a, tìm các X_i sao cho thoả mãn (*).
- Sơ đồ này thể hiện một hàm one-way với việc sinh mã rất dễ dàng nhưng việc giải mã là rất khó ➔ cơ sở xây dựng một trapdoor

Hệ PKC Merkle - Hellman

- Merkle sử dụng một mẹo là áp dụng một vector mang đặc biệt là vector siêu tăng (super-increasing)
  - thành phần i+1 là lớn hơn tổng giá trị của các thành phần đứng trước nó (1÷i).
- Việc giải mã có thể diễn ra dễ dàng như ví dụ bằng số sau:

  Vector mang siêu tăng: a=(1,2,4,8)

  Cho T=11, ta sẽ thấy việc tìm $X=(X_1,X_2,X_3,X_4)$ sao cho $T=\sum a_iX_i$ là dễ dàng:

  Đặt $T=T_0$

$X_4=1$	$T_0=T_0-X_4=3$	➔ $(X_1\ X_2\ X_3\ 1)$
$X_3=0$	$T_2=T_1=3$	➔ $(X_1\ X_2\ 0\ 1)$
$X_2=1$	$T_3=T_2-2=1$	➔ $(X_1\ 1\ 0\ 1)$
$X_1=1$		➔ $(1\ 1\ 0\ 1)$

Hệ PKC Merkle - Hellman

- Bài toán được giải quyết dần qua các bước.
  - Ở bước i, tổng đích là T_i (tức là phải tìm các a_j để tổng bằng T_i). Ta đem so sánh T_i với thành phần lớn nhất trong phần còn lại của vector, nếu lớn hơn thì thành phần này được chọn tức là X_i tương ứng bằng 1, còn ngược lại thì X_i tương ứng bằng 0. Sau đó tiếp tục chuyển sang bước sau với $T_{i+1} = T_i - X_i$.
- Cần chủ động "nguỵ trang" vector siêu tăng để chỉ có người chủ mới biết còn người ngoài không thể giải mã được.

Hệ PKC Merkle – Hellman: Cơ chế nguỵ trang

- *Tạo khoá*:

  Alice chọn một vector siêu tăng:

  $$a' = (a_1', a_2', \ldots, a_n')$$

  a' được giữ bí mật tức là một thành phần của khoá bí mật

  - Sau đó chọn một số nguyên $m > \sum a_i'$, gọi là mo-dul đồng dư và một số nguyên ngẫu nhiên ω, gọi là nhân tử, sao cho nguyên tố cùng nhau với *m*.
  - Khoá công khai của Alice sẽ là vector *a* là tích của *a'* với nhân tử ω:

    $$a = (a_1, a_2, \ldots, a_n)$$

    $$a_i = \omega \times a_i' \pmod m; \; i=1,2,3\ldots n$$

  - Còn khoá bí mật sẽ là bộ ba (a', m, ω)

Sơ đồ cụ thể Merkle-Hellman dựa trên bài toán đóng thùng.

- *Sinh mã:*
  - Khi Bob muốn gửi một thông báo X cho Alice, anh ta tính mã theo công thức:

$$T=\sum a_i X_i$$

- *Giải mã*:
  - Alice nhận được T, giải mã như sau:

Để bỏ lớp nguỵ trang cô ta trước hết tính ω^{-1} (là giá trị nghịch đảo của ω, tức là $\omega \times \omega^{-1} = 1 \bmod m$, sẽ giới thiệu thuật toán tính sau), rồi tính $T' = T \times \omega^{-1} \pmod m$

  - Alice biết rằng $T' = a'.X$ nên cô ta có thể dễ dàng giải ra được X theo siêu tăng a'.

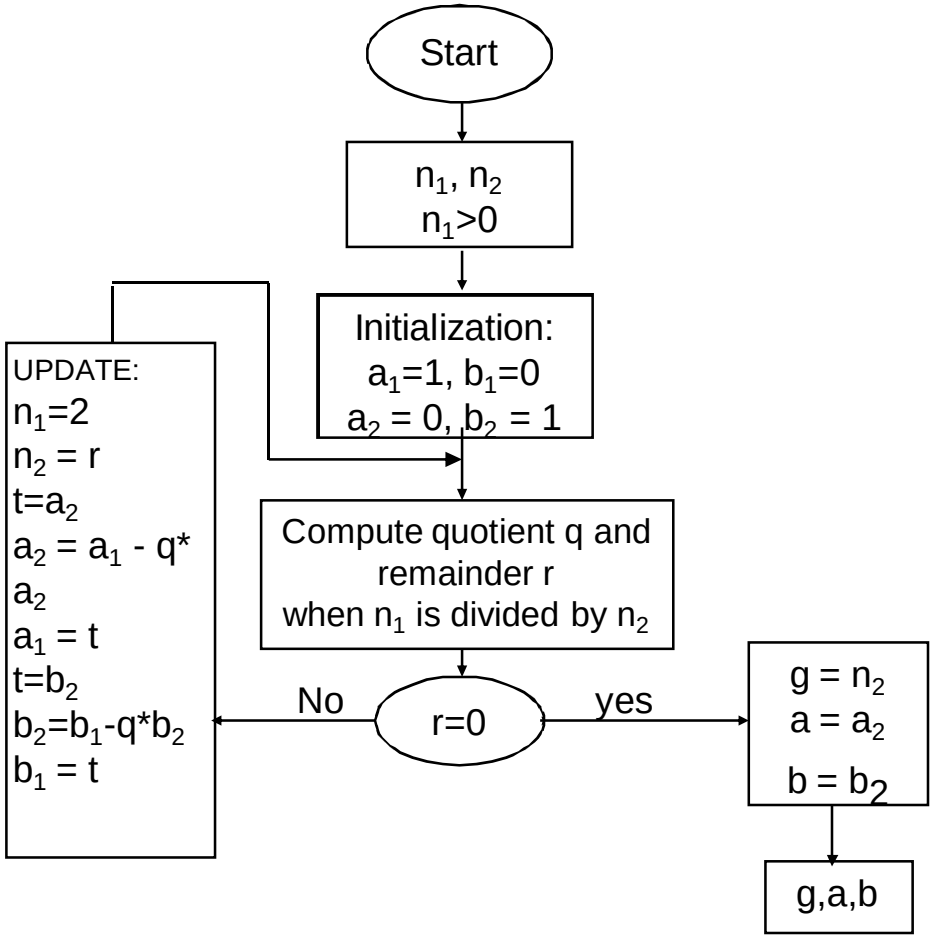
- Chú thích: ở đây ta có

$$T' = T \times \omega^{-1} = \sum a_i X_i \omega^{-1} = \sum a_i' \omega X_i \omega^{-1}$$
$$= \sum (a_i' \omega \omega^{-1}) X_i \omega^{-1} = \sum a_i' X_i = a'.X$$

- **Brute Force Attack (tấn công vũ phu)**
  - Với những kẻ không biết trapdoor (a', m, ω), giải mã đòi hỏi phải tìm kiếm vét cạn qua 2n khả năng của X.
- **Sự đổ vỡ của giải pháp dùng Knapsack (1982-1984).**
  - Shamir-Adleman đã chỉ ra chỗ yếu của GP này bằng cách đi tìm 1 cặp (ω',m') sao cho nó có thể biến đổi ngược a về a' (từ Public key về Private key).
  - 1984, Brickell tuyên bố sự đổ vỡ của hệ thống Knapsack với dung lượng tính toán khoảng 1 giờ máy Cray -1, với 40 vòng lặp chính và cỡ 100 trọng số.

Thuật toán tìm giá trị nghịch đảo theo modul đồng dư

- Việc xây dựng Knapsack với cửa bẫy đòi hỏi phải tính giá trị nghịch đảo của ω theo modul m.
- Thuật toán tìm $x = \omega^{-1} \bmod m$, sao cho $x*\omega = 1 \pmod m$ được gọi là thuật toán GCD mở rộng hay Euclide mở rộng (GCD - Greatest common divior - ước số chung lớn nhất).
  - Trong khi đi tìm ƯSCLN của hai số nguyên n_1 và n_2, người ta sẽ tính luôn các giá trị a,b sao cho $GCD(n_1, n_2) = a.n_1 + b.n_2$.
  - Từ đó suy ra nếu ta đã biết $(n_1,n_2)=1$ thì thuật toán này sẽ cho ta tìm được a, b thoả mãn $a.n_1+b.n_2=1$, tức là n_1 chính là nghịch đảo của a theo modulo n_2 (tức là m)



- Ví dụ tính bằng số: Tìm ngịch đảo của 11 theo modulo 39
- Đặt n_1=39, n_2=11 ta có bảng tính minh họa các bước như sau:

n_1	n_2	r	q	a_1	b_1	a_2	b_2
39	11	6	3	1	0	0	1
11	6	5	1	0	1	1	-3
6	5	1	1	1	-3	-1	4

Nhận xét chung về PKC

- Kể từ năm 1976, nhiều giải pháp cho PKC đã được nêu ra nhưng khá nhiều đã bị phá vỡ: chứng minh được là không an toàn.
- Một hệ thống PKC có thể đáp ứng 2 mục đích:
  - Bảo mật thông tin và truyền tin.
  - Chứng thực và chữ ký điện tử.
- Hai thuật toán đáp ứng các ứng dụng trên thành công nhất là RSA và El-Gamal.
- Nói chung PKC chậm, không thích hợp cho on-line encryption
  - Cần khi yêu cầu tính an toàn cao và chấp nhận tốc độ chậm.
  - Ngoài ra người ta thường sử dụng kết hợp PKC và SKC:
    - dùng PKC để tạo khóa bí mật thống nhất chung giữa hai bên truyền tin để thực hiện pha truyền tin chính bằng SKC sau đó.

RSA Public key cryptosystems

- RSA là hệ PK phổ biến và cũng đa năng nhất trong thực tế,
  - bởi Rivest, Shamir & Adleman.
  - Chuẩn bất thành văn PKC, cung cấp

  secrecy, authentication và digital signature.
- RSA dựa trên tính khó của bài toán phân tích các số lớn ra thừa số nguyên tố
  - Biết một số nguyên tố nhân chúng với nhau để thu được một hợp số là dễ còn biết hợp số, phân tích nó ra thừa số nguyên tố là khó.

Ý tưởng(Motivation)

- Ý tưởng của các nhà phát minh là gắn các thuật toán sinh mã và mã hoá với phép toán lấy luỹ thừa trên trường $Z_n = \{0,1,2,..n-1\}$.
  - Chẳng hạn, việc sinh mã cho tin X sẽ được thực hiện qua:

$$Y = X^e \pm n$$

Ký hiệu $a = b \pm n$ nghĩa là $a = b \pm k*n$ mà $a \in Z_n$ còn k = 1,2,3,..., ví dụ $7 = 33 \pm 10$

  - Còn việc giải mã:

$$X = X^d \pm n \text{ (e - encryption, d-decryption)}$$

- Do đó e và d phải được chọn sao cho

$$X^{ed} = X \pmod n$$

Hiện thực ý tưởng

- Người ta đã tìm được cách xây dựng cặp số (e,d) này trên cơ sở công thức như sau:

$$X^{\phi(n)} = 1 \ (mod\ n)$$ (định lý Ơ - le)

  - $\phi(n)$ là số các thuộc Z_n mà nguyên tố cùng nhau với n.
  - $\phi(n)$ có thể tính được khi đã biết công thức phân tích thừa số nguyên tố của n, cụ thể là nếu đã biết n = p*q (p,q là nguyên tố) thì $\phi(n) = (p-1)\ (q-1)$.

- Người ta chọn e*d sao cho chia $\phi(n)$ dư 1, hay

$$d = e^{-1} \pmod{\phi(n)},$$

  khi đó ta sẽ có điều cần thiết:

$$X^{ed} = X^{k.\phi(n)+1} = (X^{\phi(n)})^d.X = 1.X = X \pmod n$$

- Tóm lại: Nếu đã biết e và
  - Biết PTTSNT của n ➔ tìm được $d = e^{-1} \pmod{\phi(n)}$ tức $X^{ed} = X \pmod n$
  - Nếu không biết PTTSNT của n thì **rất khó**.

Thuật toán RSA

- Các tham số

  1. Chọn hai số nguyên tố lớn p và q.

     Tính n = p x q và m = ϕ(n) = (p = 1) * (q-1).

  2. Chọn e, 1$\leq$ e $\leq$ m -1, sao cho gcd (e, m) = 1.

  3. Tìm d sao cho e * d = 1 (mod m), tức là d = e^{-1} (mod m)

     - Giải theo thuật toán gcd mở rộng đã trình bày ở phần trước.

- Khóa công khai (Public key) là (e, n)
- Khoá dùng riêng (Private key) là (d, p, q)

Thuật toán RSA

Giả sử X là một khối tin gốc (plaintext), Y là một khối mã tương ứng của X, và (z_A, Z_A) là các thành phần công khai và riêng của khoá của Alice

- Mã hoá: Nếu Bob muốn gửi một thông báo mã hoá cho Alice thì anh ta chỉ việc dùng khoá công khai của Alice để thực hiện:

$$Y = E_{Z_A}(X) = X^e \pm n$$

- Giải mã: Khi Alice muốn giải mã Y, cô ta chỉ việc dùng khoá riêng $z_A = d$ để thực hiện như sau:

$$D_{z_A}(Y) = Y^d \pm n$$

Ví dụ bằng số

- Các tham số:
  - Chọn p = 11 và q = 13
  - n=11*13=143
  - m= (p-1)(q-1) =10 *12=120
  - e=37 ➔ gcd (37,120) =1
  - Sử dụng thuật toán gcd để tìm sao cho e * d =1 $\pm$ 120, ta tìm được d= 13 (e*d =481)
- Để mã hoá một xâu nhị phân
  - "bẻ" thành nhiều đoạn độ dài là u bit sao cho $2^u \leq 142$ ➔ u = 7. Mỗi đoạn như vậy biểu diễn một số nằm trong khoản 0 – 127
  - Tính mã Y theo công thức: $Y = X^e \pm 143$

  Chẳng hạn với X = (0000010) =2, ta có
- $E_Z(X) = X^{37} = 12 \pm 143$ ➔ Y= (00001100)
- Giải mã như sau: $X = D_z(Y) = 12^{13} = 2 \pm 143$

Thư mục khoá công khai

- Để tiện cho việc giao dịch trên mạng có sử dụng truyền tin mật, người ta có thể thành lập các Public Directory (thư mục khoá công khai), lưu trữ các khoá công khai của các user.
- Thư mục này được đặt tại một điểm công cộng trên mạng sao cho ai cũng có thể truy nhập tới được để lấy khoá công khai của người cần liên lạc.

User	(n,e)
Alice Bob Hua . . .	(85,23) (117,5) (4757,11) . . .

Ứng dụng thuật toán RSA

a. Bảo mật trong truyền tin (Confidentiality)

- A sẽ gửi $E_{Z_B}(X)$ cho B; B biết Z_B nên có thể dễ dàng giải mã.

b. Chức thực

- Alice ký lên tin cần gửi bằng cách mã hoá với khoá bí mật của cô ta $D_{z_A}(X)$ và gửi $(X,S)=(X,D_{z_A}(X))$ cho Bob
- Khi Bob muốn kiểm tra tính tin cậy của tin nhận được, anh ta chỉ việc tính $X'=E_{Z_A}(X)=E_{Z_A}(D_{z_A}(X))$ và kiểm tra nếu X = X' thì tức là tin nhận được là đáng tin cậy (authentic).

Ứng dụng RSA

Chú ý

- Tính toàn vẹn thông tin và danh tính người gửi được xác thực đồng thời:
  - Chỉ một bit của tin mà bị thay đổi thì sẽ lập tức bị phát hiện ngay do chữ ký không khớp
  - Không ai có thể tạo ra được thông báo đó ngoài Alice vì chỉ có duy nhất Alice biết z_A.
- Alice có thể ký vào giá trị băm (hash) của X thay vì ký thẳng lên X. Khi đó toàn bộ mã mà Alice sẽ chuyển cho Bob là $(X, D_{z_A}(H(X)))$.
  - H() là một hàm băm công khai.
  - Phương pháp này là hiệu quả hơn do tiết kiệm (hàm băm luôn cho ra một xâu độ dài cố định và thường << độ dài đầu vào.

Ứng dụng RSA

c. Kết hợp tính mật và tin cậy.

- Chúng ta có thể làm như sau để kết hợp cả hai khả năng a và b như trên.
- A gửi $Y = E_{Z_B}(D_{z_A}(X))$ cho B
- B phục hồi x như sau:

$$X = E_{Z_A}(D_{z_B}(Y)) = E_{Z_A}(D_{z_B}(E_{Z_B}(D_{z_A}(X))))$$

- Để có bằng chứng nhằm đối phó với việc Alice có thể sau này phủ nhận đã gửi thông báo (non-repudiation) thì Bob phải lưu giữ $D_{z_A}(X)$

Xung quanh thuật toán RSA

- ***Vấn đề chọn p và q:***
  - p và q phải là những số nguyên tố lớn, ít nhất là cỡ 100 chữ số.
  - p và q phải lớn cỡ xấp xỉ nhau (về độ dài cùng 100 chữ số chẳng hạn).
- ***Một vài con số về tốc độ thuật toán trong cài đặt:***
  - So sánh với DES thì RSA:
    - có tốc độ chậm hơn rất nhiều.
    - Kích thước của khoá mật lớn hơn rất nhiều.
  - Nếu p và q cỡ 300 bits thì n cỡ 600 bits. Phép nâng lên luỹ thừa là khá chậm so với n lớn, đặc biệt là nếu sử dụng phần mềm
  - Tốc độ hiện thời:
    - Sử dụng phần cứng đặc chủng: n cỡ 507 bits thì đạt được tốc độ khoảng 220Kb/s
    - Phần mềm: n cỡ 512 bits thì đạt được tốc độ khoảng 11Kb/s

Về bài toán phân tích ra thừa số nguyên tố

- Giải thuật tốt nhất vẫn là phương pháp sàng số. Một ước lượng về thời gian thực hiện của giải thuật là:

$$L(n) \approx 10^{9.7+\frac{1}{50}\log_2 n}$$

  - $\log_2 n$ cho số biết số bit cần để biểu diễn n, số cần phân tích TSNT
  - Người ta đã ước lượng thấy, với n=200, L(n) ≈ 55 ngàn năm.
  - Đối với khả năng thực hiện bằng xử lý song song, một trong các kết quả tốt nhất về phân tích TSNT với số lớn cho biết đã phân tích một số có 129 chữ số, phân bố tính toán trên toàn mạng Internet và mất trọn 3 tháng (1996-7)
  - Ngày nay, với những ứng dụng có độ đòi hỏi an toàn đặc biệt cao người ta sử dụng đại lượng modulo của RSA này lên đến 1024 bit và thậm chí 2048 bit.

Giải thuật tính luỹ thừa nhanh

- Luỹ thừa có thể được tính như thông thường bằng phép nhân liên tục tuy nhiên tốc độ sẽ chậm. Luỹ thừa trong trường Z_n (modulo n) có thể tính nhanh như sau:
- Để tính X^{α} (modul n):
- Xác định các hệ số αi trong khai triển của α trong hệ nhị phân:

  $\alpha = \alpha_0 2^0 + \alpha_1 2^1 + \alpha_2 2^2 + ... + \alpha_k 2^k$
- Dùng vòng lặp k bước để tính k giá trị $\pm$ n, với i=1,k :

$$X^2 = X \times X$$

$$X^4 = X^2 \times X^2$$

...

$$X^{2^k} = X^{2^{k-1}} \times X^{2^{k-1}}$$

- Do công thức nên ta tính được $X^{\alpha} \pm$ n bằng cách đem nhân với nhau các giá trị $X^2 \pm$ n đã tính ở bước 2 nếu như α_i tương ứng của nó là 1:

$$(X^{2^i})^{\alpha_i} = \begin{cases} 1, \alpha_i = 0 \\ X^{2^i}, \alpha_i = 1 \end{cases}$$