

Projets de TP de Cryptographie : La signature avec ombre

Robert Rolland

17 Avril 2004

e-mail : rolland@iml.univ-mrs.fr

Avertissement : Ceci est un exemple de projet de Travaux Pratiques de cryptographie. Bien sûr il ne s'agit pas là de la description du travail à faire, donnée aux étudiants. Il s'agit de l'étude de faisabilité du projet, ainsi que du relevé rapide des outils théoriques et des algorithmes permettant de comprendre le sujet et de le traiter. Il s'agit aussi de prévoir une démarche, qui va des réflexions théoriques jusqu'aux applications concrètes en passant par la mise en place d'algorithmes, leur programmation et les réflexions sur les résultats obtenus. Cette démarche est, me semble-t-il, apte éclairer le contexte des outils introduits, et à motiver les débutants dans l'apprentissage du sujet. La programmation et les tests sont faits avec le logiciel xcas [2]. J'attire l'attention des enseignants sur le grand intérêt de l'utilisation de ce logiciel dans les classes de divers niveaux. Dernière remarque : l'idée de ce projet est inspirée par une célèbre affaire de cassage de carte bancaire.

1 Présentation du problème

Nous présentons ici le **principe de la signature avec ombre** utilisée pour les cartes bancaires. Les nombres que nous construisons pseudo-aléatoirement sont évidemment différents de ceux utilisés pour la carte bancaire et d'ailleurs plus longs. La signature avec ombre **n'est que l'un des systèmes de sécurité** des cartes bancaires. Il s'applique dans le cas (de plus en plus rare) où le terminal de paiement du commerçant n'est pas connecté au réseau des banques. Dans ce cas il faut vérifier une "**cohérence interne**" de la carte pour garantir sa conformité.

Le principe est le suivant : Un système de signature RSA est mis en place. Le module n est le produit de deux grands nombres premiers p et q (gardés secrets par les concepteurs) de telle sorte que $\phi(n) = (p - 1)(q - 1)$ ne soit pas divisible

par 3. Cette dernière condition permet de travailler avec l'exposant public $e = 3$. L'exposant privé d (qui n'est bien entendu connu que des concepteurs) vérifie $ed \equiv 1 \pmod{\phi(n)}$. À chaque carte est associé un nombre I de k bits à partir duquel est construit le nombre "ombré" J de $2 * k$ bits obtenu en concaténant I avec lui-même. Autrement dit, $J = I * 2^k + I$. Ce nombre J est signé avec la clé privée d du système : les concepteurs calculent le nombre $A = J^d \pmod{n}$. Les deux nombres I et A sont stockés dans la carte, dans une zone accessible en lecture, après déblocage par l'entrée du bon PIN-code. La vérification se fait alors de la manière suivante : Après l'entrée du bon PIN-code par le possesseur de la carte, le terminal bancaire récupère I (et donc J) et A . Il calcule $A^3 \pmod{n}$ et vérifie qu'il trouve bien J .

Que peut faire un pirate qui veut créer une carte valide vis à vis de cette protection ? Le module n est bien entendu public. Les concepteurs n'ont pas de raison de le divulguer, mais dans ce cas il peut être facilement retrouvé, soit par le calcul (comme on va le voir dans la suite), soit par examen du programme d'un terminal de paiement. Si le pirate ne sait pas factoriser n (ce qui devrait être le cas pourvu que n soit choisi assez grand), il peut penser à partir à l'envers : Il choisit un A , calcule $J = A^3 \pmod{n}$. La probabilité pour qu'il tombe sur un nombre J "ombré" est infime.

2 Mise en place du système

Nous allons simuler un tel système en utilisant un logiciel de calcul. Le logiciel utilisé est **xcas** [2]

2.1 Le module

Nous allons construire deux nombres premiers p et q de 512 bits chacun au (pseudo) hasard. Pour construire p on construit au hasard un nombre a de 512 bits et on utilise la fonction **nextprime()** appliquée à a . Pour q on refait la même chose à partir d'un autre nombre b au hasard, de 512 bits. (Pour les principes qui soutiennent la fonction **nextprime()**, se référer au cours [1] (densité des nombres premiers, tests de primalité)). On construira p et q tels que $(p - 1)$ et $(q - 1)$ ne soient pas divisibles par 3.

2.2 Les exposants de chiffrement et de déchiffrement

Les deux nombres p et q étant construits, la valeur de e ayant été fixée à 3, il ne reste plus qu'à calculer d . Pour cela il faut résoudre l'équation de Bézout :

$$ed - k(p - 1)(q - 1) = 1.$$

La méthode standard est l'algorithme d'Euclide étendu. Cependant ici, comme $e = 3$, on sait qu'il existe une solution avec $1 \leq k \leq 2$. Il suffit donc de tester lequel des deux nombres $1 + (p - 1)(q - 1)$, $(1 + 2(p - 1)(q - 1))$ est divisible par 3. On en déduit tout de suite d .

3 Programmation des fonctions utiles

```
#####  
## alea(x)  
#####  
  
#### entree : nombre de bits x  
#### sortie : nombre aleatoire s de x bits (exactement)  
#### Algorithme de type Horner  
  
alea:=proc(x)  
  local n,s;  
  s:=1;  
  for n from 1 to x-1  
  do  
    s:=rand(2)+2*s;  
  od;  
  RETURN(s);  
end;  
  
#####  
## nbp(x)  
#####  
  
#### entree : nombre de bits x
```

```
#### sortie : nombre premier aleatoire p de x bits (exactement)
#### tel que p-1 ne soit pas divisible par 3
```

```
nbp:=proc(x)
  local p,a;
  p:=0;
  while (p=0)
    do
      a:=alea(x);
      p:=nextprime(a);
      while (irem(p-1,3)=0)
        do
          p:=nextprime(p+1);
        od;
      if p>=2^x
        then p:=0;
      fi;
    od;
  RETURN(p);
end;
```

```
#####
## RSA(x)
#####
```

```
#### entree : nombre de bits x de p et de q
#### sortie : u:=[p,q,n,f,ee,d]
#### p et q sont deux nombres premiers aléatoires de x bits
#### n est le produit pq , f est le produit (p-1)(q-1)
#### non divisible par 3,
#### ee est l'exposant public 3, d est l'exposant prive.
```

```
rsa:=proc(x)
  local u,p,q,n,f,ee,d;
  p:=nbp(x);
  q:=nbp(x);
  n:=p*q;
  f:=(p-1)*(q-1);
```

```

ee:=3;
if irem(1+f,3)=0
  then d:=iquo(1+f,3);
  else d:=iquo(1+2*f,3);
fi;
u:=[p,q,n,f,ee,d];
RETURN(u);
end;

#####
## J(x)
#####

#### entree : nombre de bits x
#### sortie : m un nombre aleatoire avec ombre, de 2x bits.

J:=proc(x)
  local m;
  m:=alea(x);
  m:=2^x *m+m;
  RETURN(m);
end;

```

4 Une session de calcul

On a reproduit ici les étapes importantes de la session.

Calcul d'un nombre m ombré.

$$m := \text{of}(J, [160]) =$$

191381212860818595173073621977680943775371118412
8845556749405056934328317886453161960069974523015

Vérification que m est bien ombré.

$$\text{iquo}(m, 2^{160}) =$$

1309483396887138809521229406871667794774305602695

$$\text{irem}(m, 2^{160}) =$$

1309483396887138809521229406871667794774305602695

Mise en place du système de signature RSA.

$$u := \text{of}(\text{rsa}, [512]) = [p, q, n, f, ee, d] =$$

[22309906155339105561747258172747934211838690139636808
925028593614361911174440489568186416210127468593494617
78890377979011890602869274118155684419856272601,

130070278898976784785713626248813232487852141720639413
662376116861098013460789493376854188817989665836373072
9504721614258665921832762759063151806483801287,

290185571583495634908556285018536715734198118276482251
10772333892156536127001992452810057204914117796285004774
23579001511391307700007090338362957923746791627775352484
07728479855075680776040359617561918164109177404447324948
37236970995033787434102296850914564659322188263074195040
47064362222873780445386637487,

29018557158349563490855628501853671573419811827648225110
77233389215653612700199245281005720491411779628500477423
57900151139130770000709033836295792374678809608194796039
88807666886770448346575337851880115660626474239432023124
71645155612791559901037041573743273423823726714148947522
362325345654944219046563600,

3,

19345704772233042327237085667902447715613207885098816740
51488926143769075133466163520670480327607853085666984949
05266767426087180000472689224197194916452539738796530693
25871777924513632231050225234586743773750982826288015416
47763437075194373267358027715828848949215817809432631681
574883563769962812697709067]

Après récupération des nombres n et d dans le tableau précédent on calcule la signature de m .

$a := \text{powmod}(m, d, n) =$
22899095928567191928917700774422548400530644861426055837
91415063967183317929221785364043661268805336323205741287
85501809876060102030784245050803262581234568091205163480
28756394645799810987780067786365460607694481804303264625
67162506921693213981737812210544435878915258231972719479
30040049977555388729059039

On vérifie la signature.

$b := \text{powmod}(a, 3, n) =$
19138121286081859517307362197768094377537111841288455567
49405056934328317886453161960069974523015

$m =$
19138121286081859517307362197768094377537111841288455567
49405056934328317886453161960069974523015

On vérifie que $m = b$.

5 Détermination du module

5.1 Principe

Le module, quoique public dans ce système, n'a pas besoin d'être fourni aux utilisateurs. Mais dans ce cas il est facile de le retrouver par observation de deux transactions avec deux cartes distinctes. En effet si on connaît deux couples (J_1, A_1) , (J_2, A_2) on peut écrire que :

$$A_1^3 - J_1 = k_1 n,$$

$$A_2^3 - J_2 = k_2 n.$$

En conséquence,

$$\gcd(A_1^3 - J_1, A_2^3 - J_2) = n \gcd(k_1, k_2).$$

Mais la probabilité pour que les nombres k_1, k_2 , qu'on peut supposer pris au hasard, aient un gcd supérieur à s est de l'ordre de $1/s^2$. En conséquence on calcule $\gcd(A_1^3 - J_1, A_2^3 - J_2)$, on en cherche les diviseurs < 1000 (par exemple). Après simplification par ces nombres, on obtient très probablement n .

5.2 Fonctions utiles

```
#####  
## recup(x)  
#####  
  
#### entree : deux transaction : J1, A1, J2, A2  
#### sortie : le module n  
  
recupn:=proc(J1,A1,J2,A2)  
  local a,b,i;  
  a:=A1^3-J1;  
  b:=A2^3-J2;  
  d:=gcd(a,b);  
  for i from 2 to 1000  
  do
```

```

    if irem(d,i)=0
      then d:=iquo(d,i);
    fi;
  od;
RETURN(d);
end;

```

5.3 Un session de calcul

On met en place le système de signature RSA.

$$u := \text{of}(\text{rsa}, [512]) = [p, q, n, f, ee, d] =$$

[115575051266682630132480239967727278608
 6180528445277956958783147663504878096994
 3565287164525899384315131063140185642263
 415978464285913933254198607716241327,

 1298798676767627210194987643521609081364
 3650212656148288042839836346078501707535
 5101925639817521670476490060293355850432
 27423883557347382786263908859127407,

 1501087236525180771002493004866838159709
 4004851340369768340978551089436486027532
 0914369394851882605495302846809417226345
 1743792866949436743579956877554291327026
 1406940237080569250713514671725872905221
 6625811035983987199056681889790739096064
 1333198734726963899437574877054191720015

42296637496548390519051749089,

1501087236525180771002493004866838159709
4004851340369768340978551089436486027532
0914369394851882605495302846809417226345
1743792866949436743579956877554291081571
2217505783569049460670315789858422359671
9516883178353315886075554607113260020584
4048122219213336098745879664781125285991
94453376180507928002476380356,

3,

1000724824350120514001662003244558773139
6003234226913178893985700726290990685021
3942912929901255070330201897872944817563
4495861911299624495719971251702860721047
4811670522379366307113543859905614906447
9677922118902210590717036404742173347056
2698748146142224065830586443187416857327
96302250787005285334984253571]

En voici les valeurs du module et de l'exposant privé.

$$n := \text{at}(u, 3) =$$

1501087236525180771002493004866838159709
4004851340369768340978551089436486027532
0914369394851882605495302846809417226345

1743792866949436743579956877554291327026
1406940237080569250713514671725872905221
6625811035983987199056681889790739096064
1333198734726963899437574877054191720015
42296637496548390519051749089

$$d := \text{at}(u, 5) =$$

1000724824350120514001662003244558773139
6003234226913178893985700726290990685021
3942912929901255070330201897872944817563
4495861911299624495719971251702860721047
4811670522379366307113543859905614906447
9677922118902210590717036404742173347056
2698748146142224065830586443187416857327
96302250787005285334984253571

On simule deux cartes (m_1, a_1) et (m_2, a_2) (m_i est la valeur ombrée et a_i la valeur obtenue en signant m_i).

$$m1 := \text{of}(J, [160]) =$$

1883698333446792683644936570061898120028
6224559288688284026505628428892013435941
05189313746075790

$$a1 := \text{powmod}(m1, d, n) =$$

1782609113328797750877243758176430935878
6568924280308435563386880783599430557637
0129474750904334238736731925682618623434

9335465629685690953929412180882023806662
7927931064362897548911140981255454623320
7145419125047895039351815414911149238408
2735385755273414717612631776263957857126
8879164273408497975390973548

$m2 := \text{of}(J, [160]) =$

1840243020753896572078092756907043162576
7394421600930851274760396250107282004677
13820228252686350

$a2 := \text{powmod}(m2, d, n) =$

1411747878125505580993699763752515305378
2947154876899570264397187858138418477195
3295947666408330068026135316690352454327
8863136431339736841065121975839408941281
0759543302162131474948270999852785240019
6863866896788451885861844146688711866966
1634609112194492317474446409797700282256
49102925018599070785290361680

$r := \text{of}(\text{recupn}, [m1, a1, m2, a2]) =$

1501087236525180771002493004866838159709
4004851340369768340978551089436486027532
0914369394851882605495302846809417226345
1743792866949436743579956877554291327026
1406940237080569250713514671725872905221

6625811035983987199056681889790739096064

1333198734726963899437574877054191720015

42296637496548390519051749089

On constate que $r=n$. On a donc bien récupéré le module.

Références

- [1] **Pierre Barthélemy, Robert Rolland, Pascal Véron.** *Cours de Cryptographie*. En cours de rédaction
- [2] **Bernard Parisse, Renée De Graeve.** *Paquetage logiciel Xcas*. [http ://www-fourier.ujf-grenoble.fr/~parisse/giac_fr.html](http://www-fourier.ujf-grenoble.fr/~parisse/giac_fr.html)