NOTES SUR LES ARMURES ASCII EN GPG

par

Ainigmatias Cruptos

Résumé. — Nous donnons dans cette note le fonctionnement des armures en OpenPGP.

1. Armure

1.1. Présentation. — Les diverses données qu'on est amené à manipuler sont en général des suites d'octets : clés publiques, clés privées clés secrètes etc. Ces suites d'octets, afin de pouvoir être communiquées, sont parfois transformées en caractères imprimables et entourées d'un en-tête et d'une fin.

1.2. L'encodage Base64. — Afin de transformer une suite d'octets en une suite de caractères imprimables et représentables en ASCII non étendu, on transforme 3 octets (24 bits) en 4 tranches de 6 bits. Une tranche de 6 bits représente un nombre Val vérifiant :

$$0 \leq Val < 64.$$

À chaque valeur Val on fait correspondre un symbole $Symb$ suivant la table 1.

Mots clefs. — cryptographie, protocole cryptographique, pgp, gpg, signature, chiffrement, crc-24, base64, radix-64.

Val	Symb	val	Symb	Val	Symb	Val	Symb
0	A	16	Q	32	g	48	w
1	B	17	R	33	h	49	x
2	C	18	S	34	i	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
6	G	22	W	38	m	54	2
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
10	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	+
15	P	31	f	47	v	63	/

TABLE 1. Table de conversion

Comme le texte d'entrée n'a pas forcément un multiple de 3 octets, la fin du texte de sortie doit être précisée.

- Si le texte d'entrée a $3k$ octets, tout se passe bien on les regroupe en exactement $4k$ blocs de 6 bits qui sont traduits en $4k$ symboles suivant la table 1.
- Si le texte d'entrée a $3k + 1$ octets, alors on rajoute 4 bits nuls à la suite des 2 bits restés seuls après regroupement par paquets de 6 bits. Les 6 bits ainsi obtenus sont transformés en 1 symbole suivant la table 1, symbole qui bien sûr est mis dans le texte de sortie, et de plus, on rajoute à la fin du texte de sortie 2 signes =.
- Si le texte d'entrée a $3k + 2$ octets, alors on rajoute 2 bits nuls à la suite des 4 bits restés seuls après regroupement par paquets de 6 bits. Les 6 bits ainsi obtenus sont transformés en 1 symbole suivant la table 1, symbole qui bien sûr est mis dans le texte de sortie, et de plus, on rajoute à la fin du texte de sortie 1 signe =.

Remarque : le nombre de symboles du texte de sortie, y compris les éventuels signes =, est un multiple de 4.

1.3. Le contrôle CRC-24. — Nous expliquons ici comment on calcule à partir de données binaires (par exemple le fichier binaire d'une clé publique) un code de contrôle (ici le CRC-24). Les données pour lesquelles on veut calculer un code de contrôle CRC (Cyclic Redondancy Code) sont considérées comme une suite d'octets :

$$V_0, V_1, \cdots, V_{s-1}.$$

L'octet V_i est lui même constitué de 8 bits suivant le format :

$$V_i = b_7^{(i)} b_6^{(i)} b_5^{(i)} b_4^{(i)} b_3^{(i)} b_2^{(i)} b_1^{(i)} b_0^{(i)}.$$

La concaténation :

$$M = V_0 || V_1 || \cdots || V_{s-1}$$

des octets de cette suite nous fournit une suite de $n = 8s$ bits :

$$\begin{aligned} M \;=\; & b_7^{(0)} b_6^{(0)} b_5^{(0)} b_4^{(0)} b_3^{(0)} b_2^{(0)} b_1^{(0)} b_0^{(0)} \cdots \\ & \cdots b_7^{(i)} b_6^{(i)} b_5^{(i)} b_4^{(i)} b_3^{(i)} b_2^{(i)} b_1^{(i)} b_0^{(i)} \cdots \\ & \cdots b_7^{(s-1)} b_6^{(s-1)} b_5^{(s-1)} b_4^{(s-1)} b_3^{(s-1)} b_2^{(s-1)} b_1^{(s-1)} b_0^{(s-1)}. \end{aligned}$$

Afin de simplifier un peu les notations, notons :

$$M = m_{n-1} m_{n-2} \cdots m_0$$

cette suite de bits. Ainsi :

$$m_{n-1} = b_7^{(0)},$$
$$m_{n-2} = b_6^{(0)},$$

et plus généralement :

$$m_i = b_{i \mod 8}^{\left(s - \lceil \frac{i}{8} \rceil\right)},$$

où $\lceil \frac{i}{8} \rceil$ désigne le plus petit entier $> i/8$ et $i \mod 8$ le reste de la division de i par 8.

On considère alors que M (qui est une suite de n bits) peut en fait représenter le polynôme $M(x)$ de degré $n-1$ à coefficient dans le corps à deux éléments $\{0, 1\}$, dont les coefficients sont les bits m_i :

$$M(x) = m_{n-1} x^{n-1} + m_{n-2} x^{n-2} + \cdots + m_i x^i + \cdots + m_1 x + m_0.$$

On fixe deux polynômes, le polynôme générateur :

$G(x) = x^{24}+x^{23}+x^{18}+x^{17}+x^{14}+x^{11}+x^{10}+x^7+x^6+x^5+x^4+x^3+x+1,$

et le polynôme initial : :

$I(X) = x^{23} + x^{21} + x^{20} + x^{18} + x^{17} + x^{16} + x^{10} + x^7 + x^6 + x^3 + x^2 + x.$

Si on écrit les coefficients sous forme de suites de bits on obtient :

$$G = 1100001100100110011111011,$$

$$I = 1011011100000100110011110,$$

ou encore en hexadécimal :

$$G = 0x1864cfb,$$

$$I = 0xb704ce.$$

Le calcul du crc de la donnée M se fait alors de la façon suivant :

(1) on calcule :

$$A(x) = x^{24}\, M(x) + x^n\, I(x),$$

(2) on fait la division de $A(x)$ par $G(x)$, le reste est un polynôme $R(x)$ de degré au plus 23. Les 24 bits correspondants aux coefficients de ce polynôme constituent le **crc**. On a donc 3 octets de crc.

Remarque : Les deux polynômes $x^{24}\, M(x)$ et $x^n\, I(x)$ ont le même degré. La somme des deux polynômes correspond à un ou exclusif bit à bit sur les deux suites binaires représentant leurs coefficients.

Voici un programme qui calcule le CRC-24. Il utilise la procédure *crc_octets* qui est exactement celle donnée dans la note **RFC 2440**.

```
#include <stdlib.h>
#include <math.h>


/**************************************************************************

#define CRC24_INIT 0xb704ceL
#define CRC24_POLY 0x1864cfbL
typedef long crc24;
```

```
crc24 crc_octets(unsigned char *octets, size_t len)
{
  crc24 crc=CRC24_INIT;
  int i;
  while (len--)
    {
      crc ^= (*octets++) <<16;
      for (i=0;i<8;i++)
        {
          crc <<= 1;
          if (crc & 0x1000000)
            crc^= CRC24_POLY;
        }
    }
  return crc & 0xffffffL;
}

void main()
{
  char *suite;
  size_t len=2;
  suite=(char *) malloc(len*sizeof(char));
  suite[0]=(char) 0xA5;
  suite[1]=(char) 0xca;
  crc24 crc=crc_octets((char *) suite,len);
  printf("%lx\n",crc);
}
```

1.4. Synthèse : la transformation Radix-64. — La transformation Radix-64 consiste en deux opérations. D'une part le calcul du crc. D'autre part l'utilisation de base64. Voici un exemple de clé publique (révoquée) sous forme d'armure ascii :

```
 -----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1.4.5 (GNU/Linux)
```

```
mIsEQ6TTOAEEANPVWUZar2lwmus7ziNqpX4CKcXFDqM2v+34O6RlIdgeiJdp8jgU
yTXssymZnaCidgWVw70Z+G7b+/EBhSD7J1wFXaYdbY9M44PQr5S+45oCMiylanVA
1VH4KPy4LKzaZfx4HCBHS2YuRxXwkEmQ7jmiYu6NbOZ9OGGHwm9IlfdbAAYpiLIE
IAECABwFAkOk068VHQBjb3BpZSBkZSBzYXV2ZWdhcmRlAAoJEAHVEZOOCNHdgwME
AMCblyI4g1ogsHGrm3xQtflqDOUqSQtVmy1auWUl2uWdE7KNMOiofTCK+CjpcJPn
k3meQWSbpX6LCqFaviPV9dw9umjqIOqyLZ1YPuufGGdFDjtVSaq4k66VqZEdfc6B
xRUj7ya7xaO96Txs+lsbleRiGadCDepOxrL2Q+mNZ6C+tD9Sb2JlcnQgUm9sbGFu
ZCAoYWRyZXNzZSBwZXJzb25uZWxsZSkgPFJvYmVydC5Sb2xsYW5kMTNAZnJlZS5m
cj6IswQTAQIAHQUCQ6TTOAYLCQgHAwIEFQIIAwQWAgMBAh4BAheAAAoJEAHVEZOO
CNHdqe4D/RMx6QLQ9oL3+e0+efPNpUR20uZKVUMBO7uvHRtLhUzvHR3OPIRwkMz1
dx1hZx2SZVk7ZzIEjR+ujOzyt5hheEzv5J5ShY/hZA+xJkNB8rIzqw6XctAff6RF
A5AomlNodAd5KTfUUBuQd8Guq8y1cKqTZiT7z1rreO4BszvIdMcp
=TJ6m
-----END PGP PUBLIC KEY BLOCK-----
```

On voit donc l'en-tête :

```
 -----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1.4.5 (GNU/Linux)
```

Puis la clé sous forme base64 (c'est ce qu'on obtient en transformant le fichier binaire de la clé publique par base64) :

```
mIsEQ6TTOAEEANPVWUZar2lwmus7ziNqpX4CKcXFDqM2v+34O6RlIdgeiJdp8jgU
yTXssymZnaCidgWVw70Z+G7b+/EBhSD7J1wFXaYdbY9M44PQr5S+45oCMiylanVA
1VH4KPy4LKzaZfx4HCBHS2YuRxXwkEmQ7jmiYu6NbOZ9OGGHwm9IlfdbAAYpiLIE
IAECABwFAkOk068VHQBjb3BpZSBkZSBzYXV2ZWdhcmRlAAoJEAHVEZOOCNHdgwME
AMCblyI4g1ogsHGrm3xQtflqDOUqSQtVmy1auWUl2uWdE7KNMOiofTCK+CjpcJPn
k3meQWSbpX6LCqFaviPV9dw9umjqIOqyLZ1YPuufGGdFDjtVSaq4k66VqZEdfc6B
xRUj7ya7xaO96Txs+lsbleRiGadCDepOxrL2Q+mNZ6C+tD9Sb2JlcnQgUm9sbGFu
ZCAoYWRyZXNzZSBwZXJzb25uZWxsZSkgPFJvYmVydC5Sb2xsYW5kMTNAZnJlZS5m
cj6IswQTAQIAHQUCQ6TTOAYLCQgHAwIEFQIIAwQWAgMBAh4BAheAAAoJEAHVEZOO
CNHdqe4D/RMx6QLQ9oL3+e0+efPNpUR20uZKVUMBO7uvHRtLhUzvHR3OPIRwkMz1
dx1hZx2SZVk7ZzIEjR+ujOzyt5hheEzv5J5ShY/hZA+xJkNB8rIzqw6XctAff6RF
A5AomlNodAd5KTfUUBuQd8Guq8y1cKqTZiT7z1rreO4BszvIdMcp
```

On rajoute un nouvelle ligne commençant par le signe ′ =′ suivi des 4 caractères formant le base64 du CRC-24 du fichier binaire de la clé publique :

```
=TJ6m
```

Et enfin la fin :

```
-----END PGP PUBLIC KEY BLOCK-----
```

---

27 février 2012

A. Cruptos, Association ACrypTA. • *E-mail :* acrypta@acrypta.fr