Cryptographie et protocoles cryptographiques

Jean Goubault-Larrecq

LSV/CNRS UMR 8643, ENS Cachan

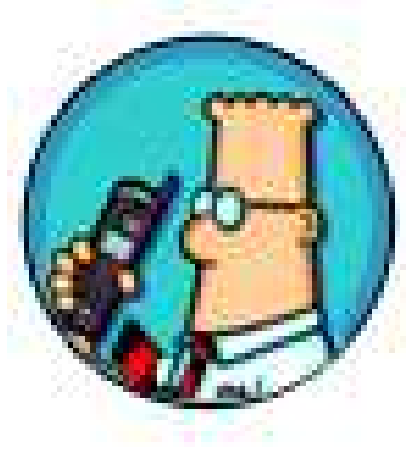
Laboratoire Spécification et Vérification

Plan

1. Comment envoyer des messages secrets ?
2. Un aperçu de quelques algorithmes de chiffrement.
3. Quelques autres propriétés de sécurité.
4. Comment tirer un nombre au hasard ?
5. Les protocoles cryptographiques ; SSL.

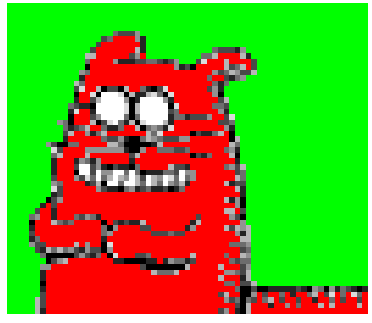
Alice veut envoyer un message à Bob...

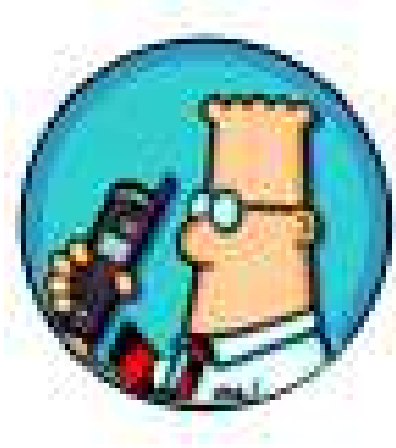
solution : 2 ; 3 ; 1 ; 1



Alice veut envoyer un message à Bob...

solution : 2 ; 3 ; 1 ; 1





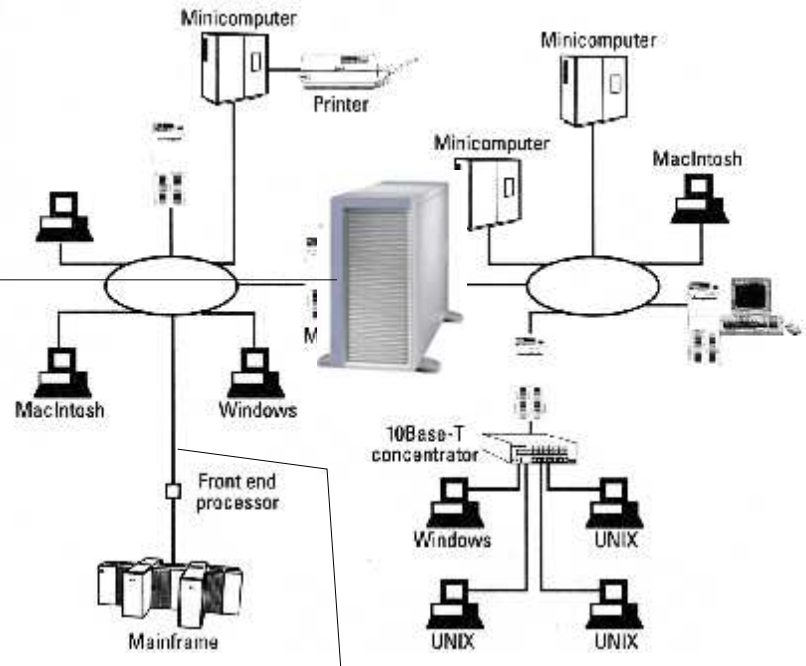
Mais Charlie peut espionner le message : pas de secret.

Concrètement

Tout ce qu'Alice tape circule en clair sur le cable qui relie son ordinateur (à gauche) au serveur (à droite).

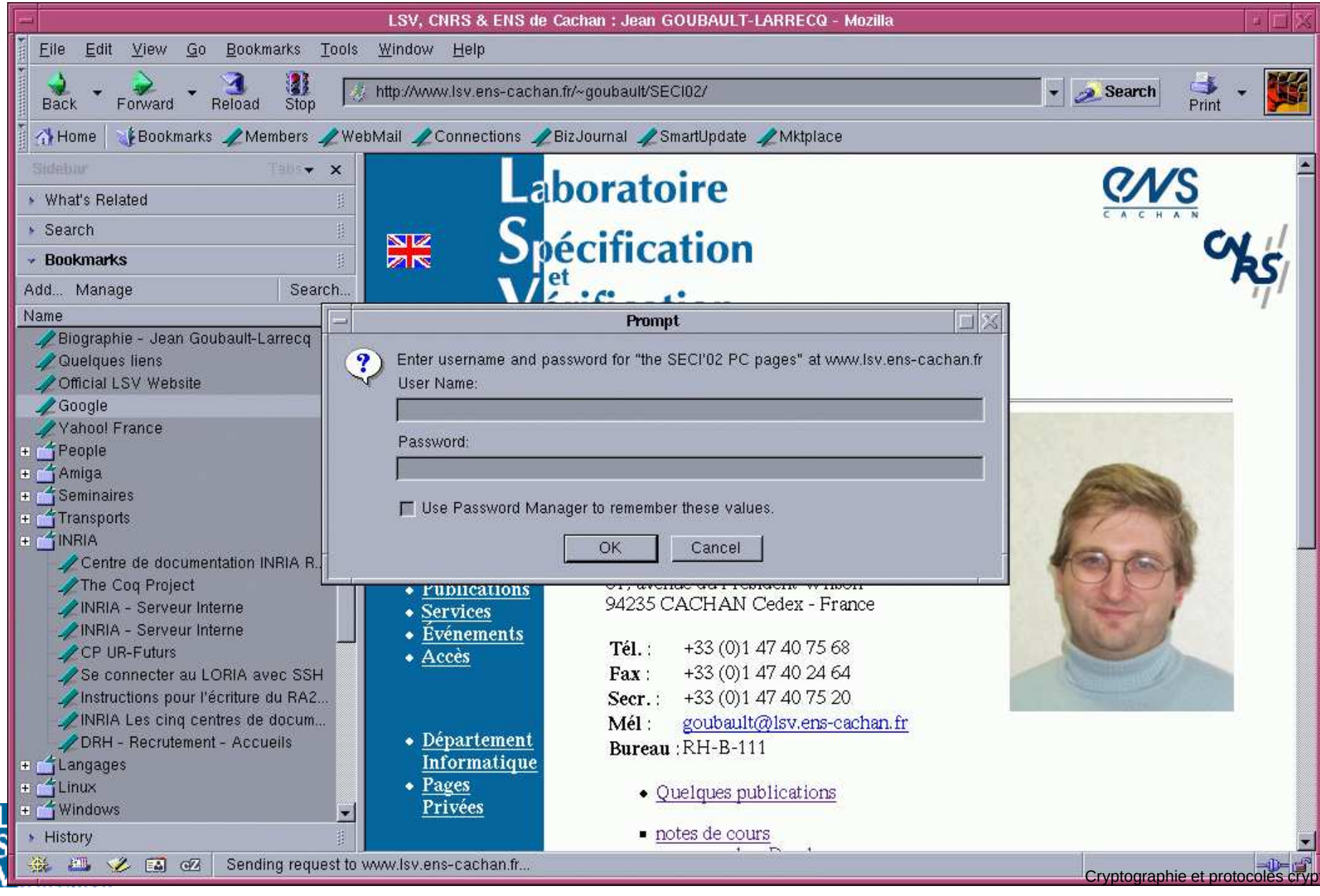
N’importe qui peut espionner !

"passwd: toto"



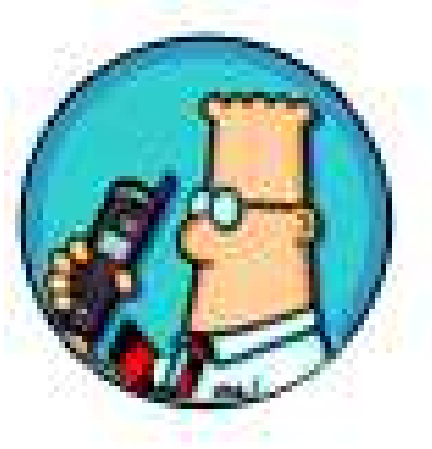
"passwd:
toto"

...même quand vous croyez que c'est sûr



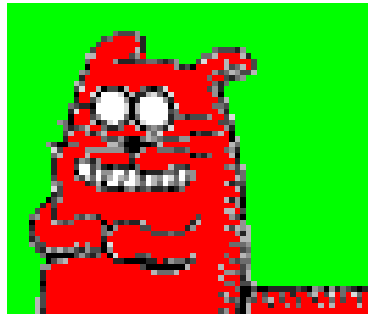
La solution standard : le chiffrement

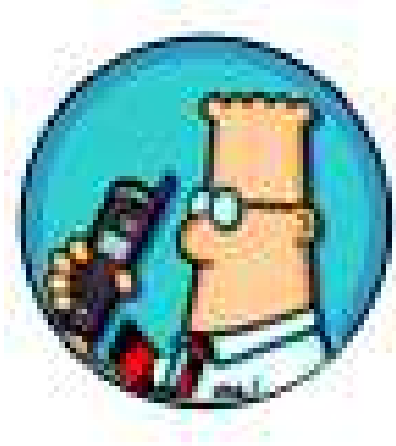
vroxwlrq = 5> 6> 4> 4



La solution standard : le chiffrement

vroxwlrq = 5> 6> 4> 4





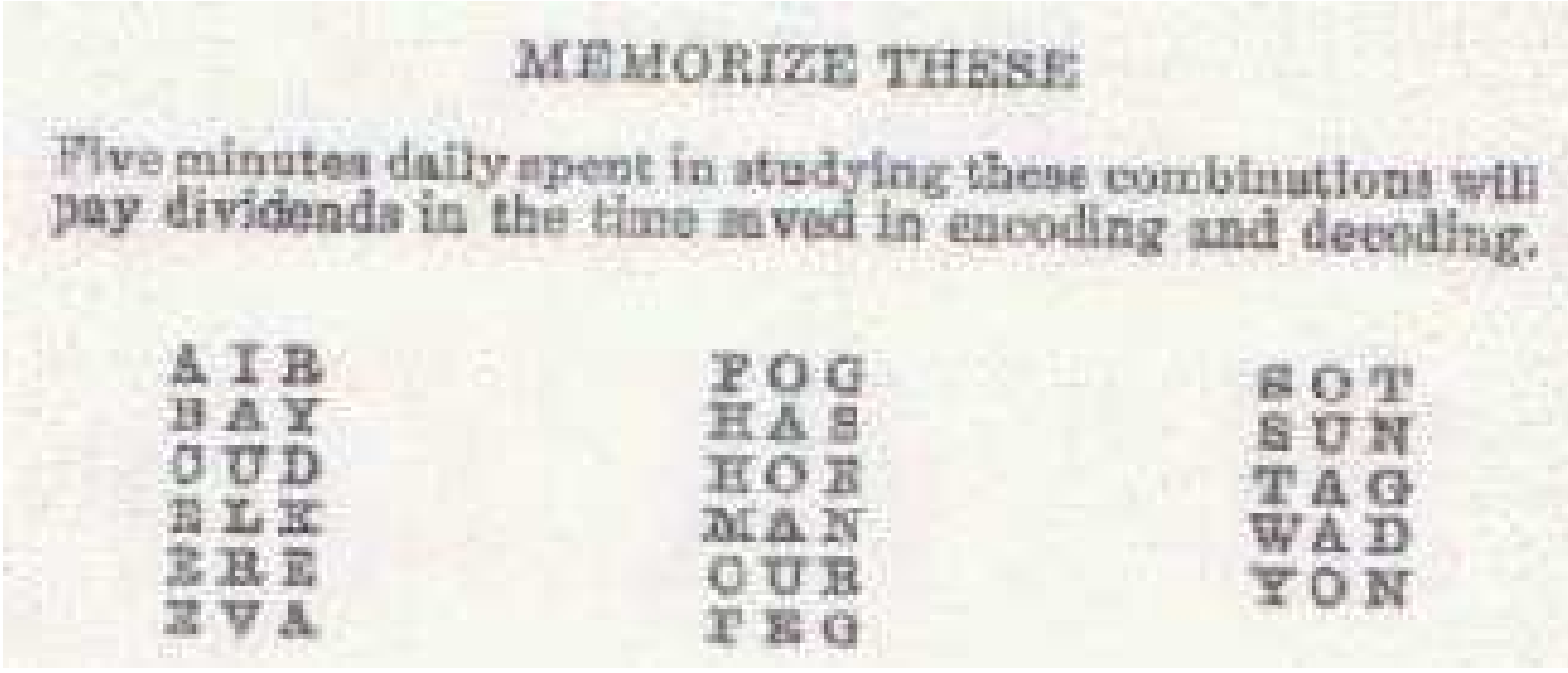
Mais l'algorithme de chiffrement est trop simple...

Plan

1. Comment envoyer des messages secrets ?
2. Un aperçu de quelques algorithmes de chiffrement.
3. Quelques autres propriétés de sécurité.
4. Comment tirer un nombre au hasard ?
5. Les protocoles cryptographiques ; SSL.

Un meilleur système de chiffrement

Envoyer $k$ième lettre $\oplus$ $k$ième lettre d'un masque jetable (one-time pad), $1 \leq k \leq$ longueur :
($\oplus$ =ou exclusif, ou bien addition mod 26, etc.)



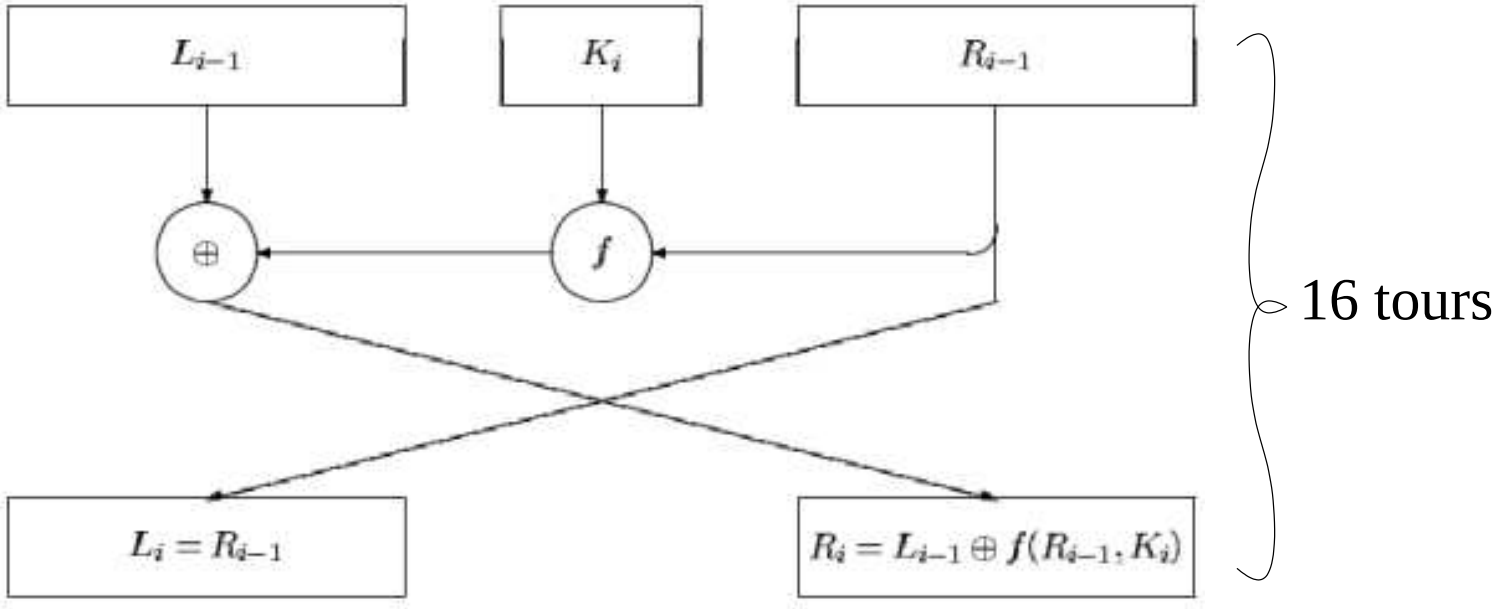
MEMORIZE THESE

Five minutes daily spent in studying these combinations will pay dividends in the time saved in encoding and decoding.

A I B	P O G	S O T
B A Y	H A S	S U N
O U D	H O E	T A G
N L K	M A N	W A D
E H E	O U R	Y O N
E V A	P E G	

(Celui-ci est bien entendu ridicule...)

Et il y en a d'autres, le DES par exemple...



C'est un cas particulier de chiffrement de Feistel.

Un petit point de vocabulaire

En Français, on dit :

– chiffrer, pas “crypter” (to encrypt)

– déchiffrer, pas “décrypter” (to decrypt)

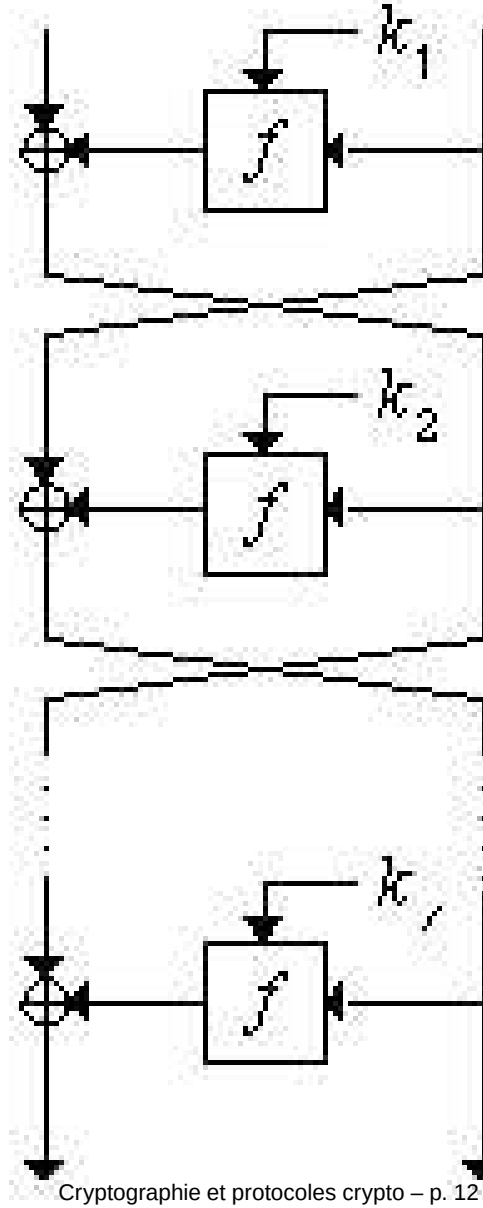
Un petit point de vocabulaire

En Français, on dit :

– chiffrer, pas “crypter” (to encrypt)
– déchiffrer, pas “décrypter” (to decrypt)
– décrypter... (to decrypt, to crack)

Les chiffrements de Feistel

```
proc Feistel (L, R : int)
     for i = 1 to N do
         R := L xor f (R, k[i]);
         L := R;
     return (R, L);
```



Les chiffrements de Feistel

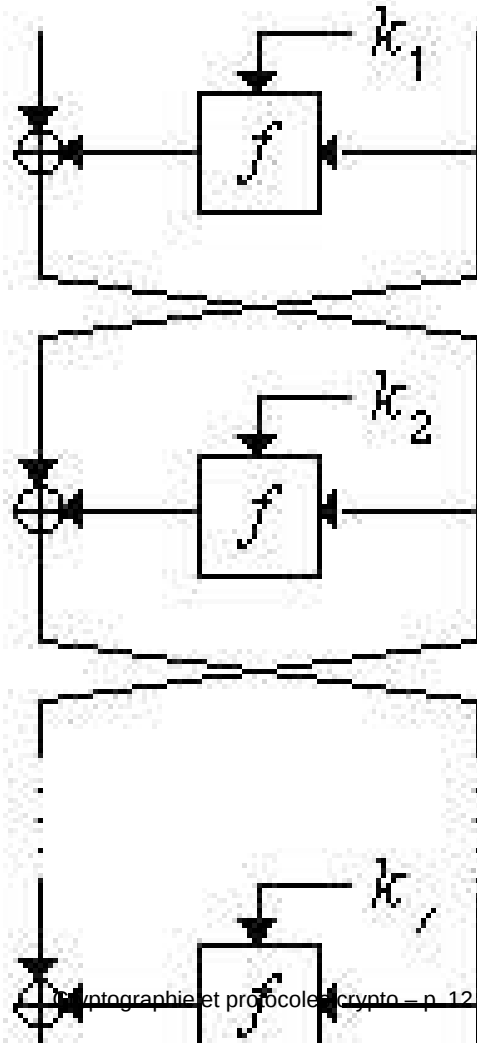
```
proc Feistel (L, R : int)
     for i = 1 to N do
         R := L xor f (R, k[i]);
         L := R;
     return (R, L);
```

Pour le DES : $N = 16$, L et R sur 32 bits, f est calculée à l'aide de tables (les S-box).

Le tableau `k[i]` est obtenu par un processus d'expansion à partir d'une clé de 64 bits (56 bits effectifs, 8 bits de checksum).

Les chiffrements de Feistel sont très rapides à calculer... surtout en hardware (circuits DES).

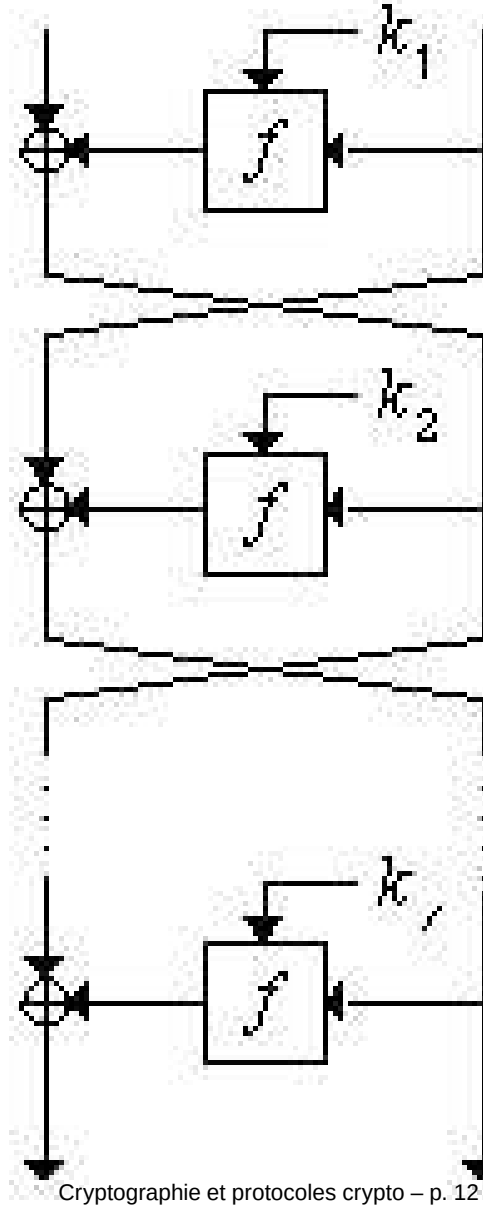
Et déchiffrer est très facile !



Laboratoire Spécification et Vérification

Les chiffrements de Feistel

```
proc Feistel (L, R : int)
     for i = 1 to N do
         R := L xor f (R, k[i]);
         L := R;
     return (R, L);
proc Feistel-inv (R, L : int)
     for i = 1 to N do
         R := L xor f (R, k[N+1-i]);
         L := R;
     return (L, R);
```



Laboratoire Spécification et Vérification

Un chiffrement voisin : RC5 (RFC2040)

```
proc RC5 (L, R : int)
     L := L + k[0]; R := R + k[1];
     for i = 1 to N do
         L := rotl (L xor R, R) + k[2*i];
         R := rotl (R xor L, L) + k[2*i+1];
     return (L, R);
```

Typiquement $N = 12$ (pour une clé de 64 bits), $N = 16$ (clé de 128 bits).
L et R sont sur 32 bits.
Tableau `k[i]` calculé par expansion de clé 64 ou 128 bits.
Toujours très rapide ! ... et moins sensible aux attaques que DES (la cryptanalyse différentielle de Shamir).

IDEA (Intl. Data Encryption Algo.)

```
proc IDEA (p : int16[4])
     for i = 1 to 8 do
         d0 :=p[0]*k[6*i-6]; d1 :=p[1]+k[6*i-5];
         d2 :=p[2]+k[6*i-4]; d3 :=p[3]*k[6*i-3];
         d4 :=d0 xor d2; d5 :=d1 xor d3;
         d6 :=d4*k[6*i-2]; d7 :=d5+d6;
         d8 :=d7*k[6*i-1]; d9 :=d6+d8;
         p[0] :=d0 xor d8; p[2] :=d2 xor d8;
         p[1] :=d1 xor d9; p[3] :=d3 xor d9;
     p[0] :=p[0]*k[48]; p[1] :=p[1]+k[49];
     p[2] :=p[2]+k[50]; p[3] :=p[3]*k[51];
     return p;
```

(où l'addition est mod 2^{16}, et la multiplication mod $2^{16}+1$...)

Plan

1. Comment envoyer des messages secrets ?
2. Un aperçu de quelques algorithmes de chiffrement.
3. Quelques autres propriétés de sécurité.
4. Comment tirer un nombre au hasard ?
5. Les protocoles cryptographiques ; SSL.

Avec un chiffrement amélioré...

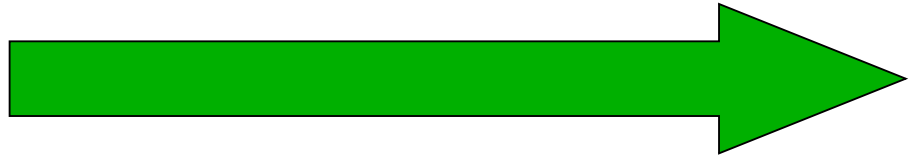
sw2z7o61 ; JB EM >= 4

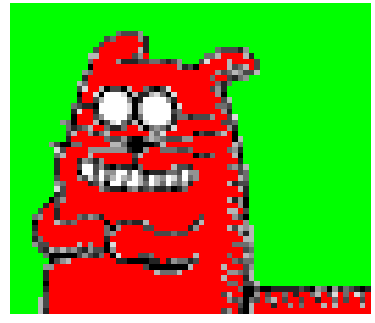
$= \{\text{solution} : 2 ; 3 ; 1 ; 1\}_K$

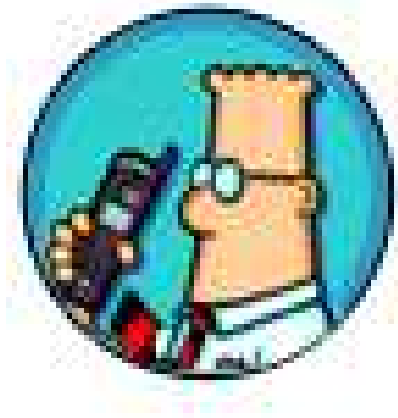
... Charlie peut encore nuire !

sw2z7o61 ; IB DM @= 4

$= \{\text{solution} : 1 ; 2 ; 3 ; 1\}_K$







L'intégrité du message est compromise.

$\Rightarrow$ éviter le chiffrement par blocs

Par ex., CBC : Découper le message en blocs (de 64 bits pour DES par ex.) :

B_1	B_2	...	B_n

Envoyer :

IV_0	$\{IV_0 \oplus B_1\}_K$ $= IV_1$	$\{IV_1 \oplus B_2\}_K$ $= IV_2$	...	$\{IV_{n-1} \oplus B_n\}_K$ $= IV_n$

Empêche de tronquer ou remplacer des bouts.

$\Rightarrow$ éviter le chiffrement par blocs

Par ex., CBC : Découper le message en blocs (de 64 bits pour DES par ex.) :

B_1	B_2	...	B_n

Envoyer :

IV_0	$\{IV_0 \oplus B_1\}_K$ $= IV_1$	$\{IV_1 \oplus B_2\}_K$ $= IV_2$	...	$\{IV_{n-1} \oplus B_n\}_K$ $= IV_n$

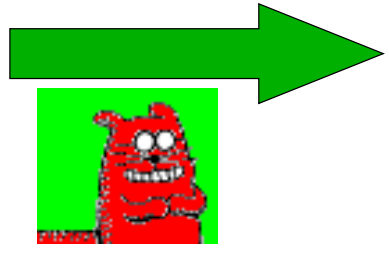
Empêche de tronquer ou remplacer des bouts.

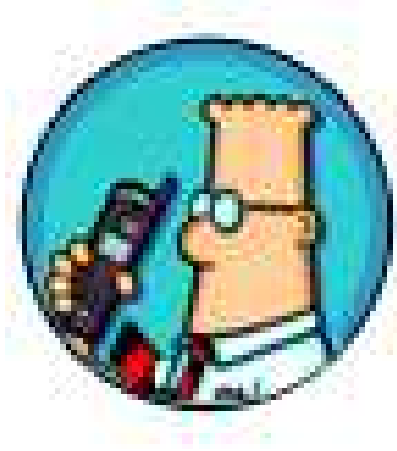
Intégrité : inclure des sommes de contrôle (fonctions de hachage ; la plus connue : MD5).

On n'arrête pas Charlie !

jm ss :o1t =hs

= $\{$je ne sais pas$\}_K$



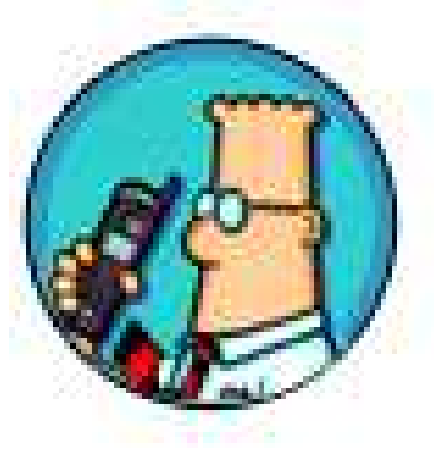


Charlie peut rejouer un vieux message, même sans connaître la clé K.

Alice inclut un temps dans son message...

sw2z7o61 ; JB EM >= 4 - C9@5=

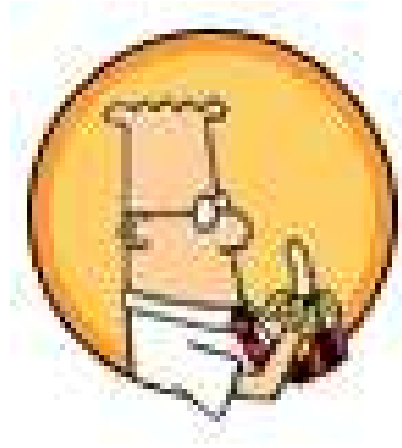
$= \{\text{solution} : 2 ; 3 ; 1 ; 1 - 09 : 12\}_K$



..Charlie ne peut plus rejouer de messages

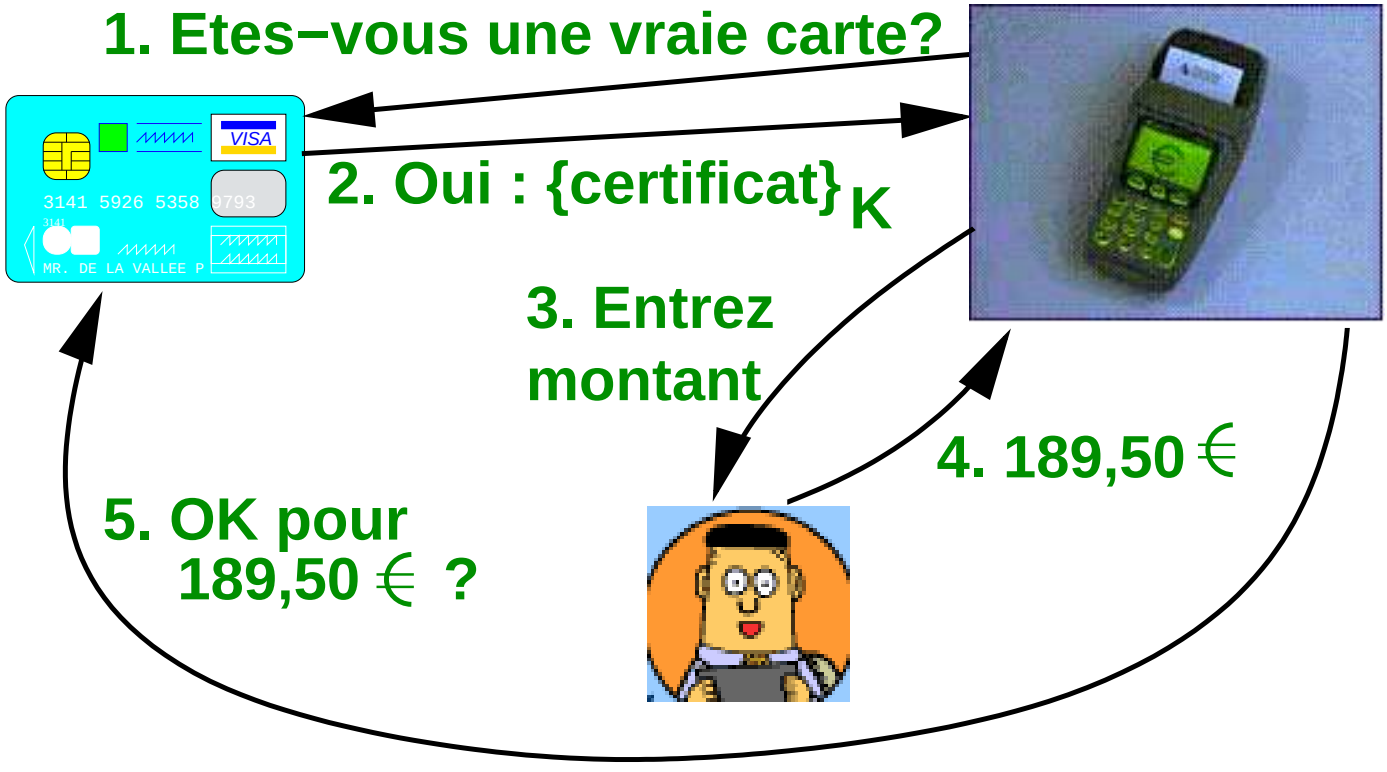
sw2z7o61 ; IB DM @= 4 - C8@4@

$= \{\text{solution} : 1 ; 2 ; 3 ; 1 - 08 :05\}_K$

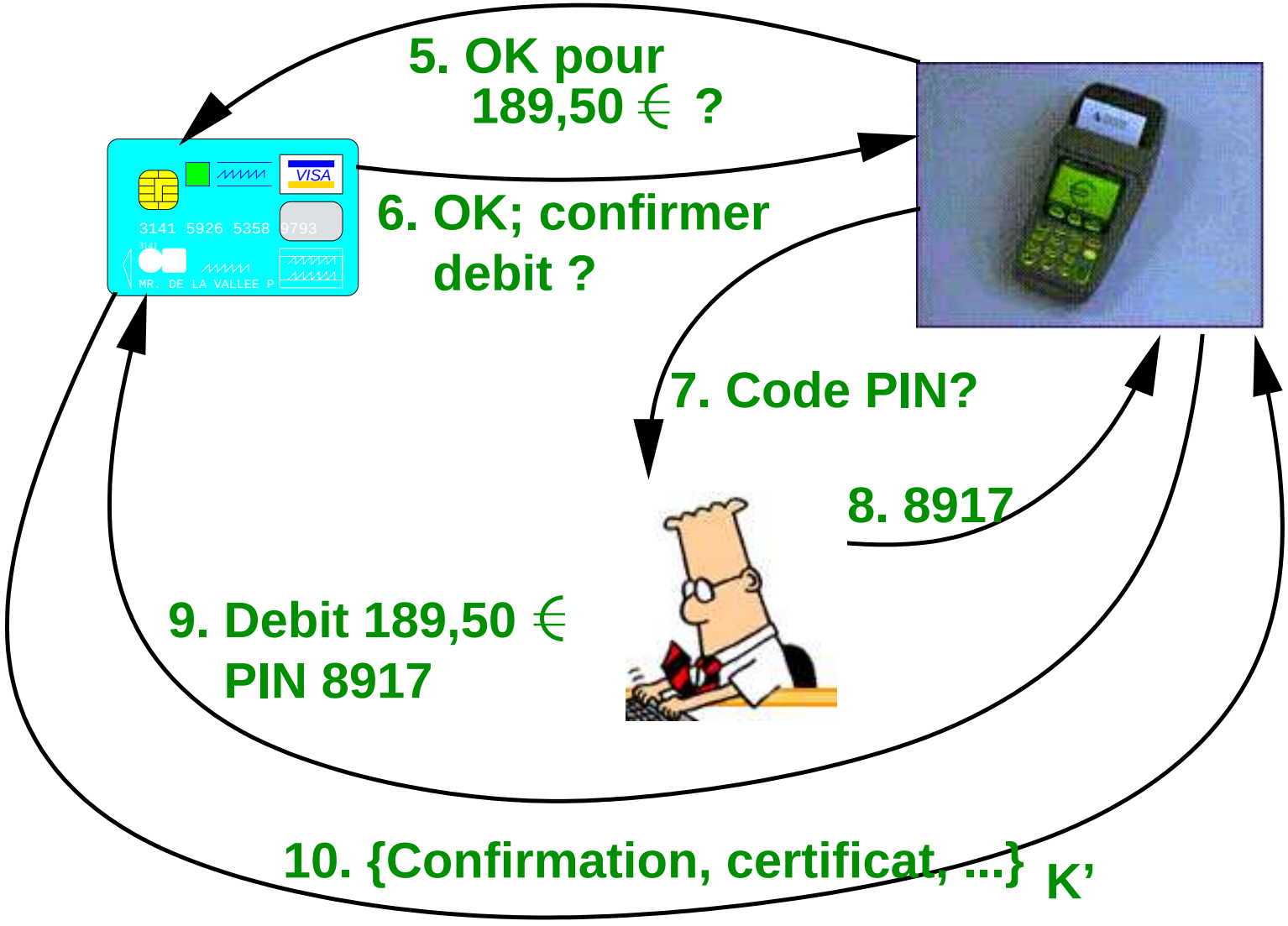


Ceci demande cependant des horloges synchronisés, donc un protocole de synchronisation (à la `ntp`) sécurisé.
On verra des solutions plus simples à base de nonces.

Un exemple concret : les cartes à puce



Les cartes à puce, suite



En résumé

Alice a dû utiliser :

- un algorithme de chiffrement fort — secret ;
- des sommes de contrôle — intégrité ;
- des estampilles temporelles — fraîcheur ;
- plus quelques autres précautions de codage (CBC, types explicites, ...) ;
- plus divers — authenticité, unicité, ...

Avec tout ça, on devrait être parés ?

Plan

1. Comment envoyer des messages secrets ?
2. Un aperçu de quelques algorithmes de chiffrement.
3. Quelques autres propriétés de sécurité.
4. Comment tirer un nombre au hasard ?
5. Les protocoles cryptographiques ; SSL.

Nonces

En fait, on a souvent besoin de fabriquer des données nouvelles (jamais vues auparavant, et imprédictibles).

Nonces

En fait, on a souvent besoin de fabriquer des données nouvelles (jamais vues auparavant, et imprédictibles).

- Utile pour fabriquer des clés nouvelles.

Nonces

En fait, on a souvent besoin de fabriquer des données nouvelles (jamais vues auparavant, et imprédictibles).

- Utile pour fabriquer des clés nouvelles.
- Peut servir à garantir la fraîcheur :

new N_a

1. $A \rightarrow B : N_a, ...$
2. $B \rightarrow A : \{N_a, ...\}_K$

garantit que le message 2 est au moins aussi récent que le message 1.

Nombres (pseudo-)aléatoires

Pour fabriquer un nombre nouveau (disons sur 128 bits), on le tire au hasard.

Nombres (pseudo-)aléatoires

Pour fabriquer un nombre nouveau (disons sur 128 bits), on le tire au hasard.

$$198\ 089\ 357\ 218\ 068\ 216\ 517\ 260\ 685\ 773\ 624\ 326\ 765$$

Nombres (pseudo-)aléatoires

Pour fabriquer un nombre nouveau (disons sur 128 bits), on le tire au hasard.

$$198\ 089\ 357\ 218\ 068\ 216\ 517\ 260\ 685\ 773\ 624\ 326\ 765$$

$$324\ 073\ 163\ 420\ 648\ 383\ 635\ 925\ 543\ 613\ 609\ 217\ 851$$

Nombres (pseudo-)aléatoires

Pour fabriquer un nombre nouveau (disons sur 128 bits), on le tire au hasard.

198 089 357 218 068 216 517 260 685 773 624 326 765

324 073 163 420 648 383 635 925 543 613 609 217 851

232 992 305 167 558 465 512 230 572 597 182 844 496

Nombres (pseudo-)aléatoires

Pour fabriquer un nombre nouveau (disons sur 128 bits), on le tire au hasard.

198 089 357 218 068 216 517 260 685 773 624 326 765
324 073 163 420 648 383 635 925 543 613 609 217 851
232 992 305 167 558 465 512 230 572 597 182 844 496
285 921 562 481 663 339 762 468 133 626 338 094 559

Nombres (pseudo-)aléatoires

Pour fabriquer un nombre nouveau (disons sur 128 bits), on le tire au hasard.

198 089 357 218 068 216 517 260 685 773 624 326 765
324 073 163 420 648 383 635 925 543 613 609 217 851
232 992 305 167 558 465 512 230 572 597 182 844 496
285 921 562 481 663 339 762 468 133 626 338 094 559
311 193 852 387 359 001 535 265 578 684 927 465 138

Nombres (pseudo-)aléatoires

Pour fabriquer un nombre nouveau (disons sur 128 bits), on le tire au hasard.

198 089 357 218 068 216 517 260 685 773 624 326 765
324 073 163 420 648 383 635 925 543 613 609 217 851
232 992 305 167 558 465 512 230 572 597 182 844 496
285 921 562 481 663 339 762 468 133 626 338 094 559
311 193 852 387 359 001 535 265 578 684 927 465 138

...

Le risque de tomber sur un nombre déjà fabriqué est infime...

Nombres pseudo-aléatoires et fraîcheur

Âge de l'univers : 15 milliards d'années $\sim 4,73\ 10^{17}$ s.

Nombres pseudo-aléatoires et fraîcheur

Âge de l'univers : 15 milliards d'années $\sim 4,73\ 10^{17}$ s.
Population totale depuis le début de l'univers $\sim 10^{10}$.

Nombres pseudo-aléatoires et fraîcheur

Âge de l'univers : 15 milliards d'années $\sim 4,73\ 10^{17}$ s.
Population totale depuis le début de l'univers $\sim 10^{10}$.
Supposons que chaque humain invente un nombre chaque seconde.

$\Rightarrow$ au plus $5\ 10^{27}$ nombres sont non frais.

Nombres pseudo-aléatoires et fraîcheur

Âge de l'univers : 15 milliards d'années $\sim 4,73\ 10^{17}$ s.
Population totale depuis le début de l'univers $\sim 10^{10}$.
Supposons que chaque humain invente un nombre chaque seconde.

$\Rightarrow$ au plus $5\ 10^{27}$ nombres sont non frais.
Il y a $3,4\ 10^{38}$ nombres de 128 bits.

Nombres pseudo-aléatoires et fraîcheur

Âge de l'univers : 15 milliards d'années $\sim 4,73\ 10^{17}$ s.
Population totale depuis le début de l'univers $\sim 10^{10}$.
Supposons que chaque humain invente un nombre chaque seconde.

$\Rightarrow$ au plus $5\ 10^{27}$ nombres sont non frais.
Il y a $3,4\ 10^{38}$ nombres de 128 bits.

On n'a donc qu'une chance sur 68 milliards (au plus !) de tirer au hasard un nombre non frais.

Aléa et imprédictibilité

Un générateur aléatoire est cryptographiquement sûr si et seulement si, étant données les n premières valeurs produites

$$x_1, x_2, ..., x_n$$

on ne peut pas produire la suivante x_{n+1} par calcul [en temps raisonnable].
... sinon un intrus pourrait produire le nonce `Na` avant `A` !

Aléa et imprédictibilité

Un générateur pseudo-aléatoire courant : le générateur linéaire congruentiel.

$$x_{n+1} = ax_n + b \bmod M$$

avec a et M premiers entre eux. (M peut être supposé connu.)

Aléa et imprédictibilité

Un générateur pseudo-aléatoire courant : le générateur linéaire congruentiel.

$$x_{n+1} = ax_n + b \bmod M$$

avec a et M premiers entre eux. (M peut être supposé connu.)
Extrêmement facile à prédire :
Connaissant x_1, x_2, et x_3,

$$a = \frac{x_2 - x_1}{x_1 - x_0} \qquad b = x_2 - ax_1$$

Maintenant l'intrus connaît x_4, x_5, etc.

Aléa et imprédictibilité

Un générateur pseudo-aléatoire courant : le générateur linéaire congruentiel.

$$x_{n+1} = ax_n + b \bmod M$$

avec a et M premiers entre eux. (M peut être supposé connu.)
Extrêmement facile à prédire :
Connaissant x_1, x_2, et x_3,

$$a = \frac{x_2 - x_1}{x_1 - x_0} \qquad b = x_2 - ax_1$$

Maintenant l'intrus connaît x_4, x_5, etc.
Malheureusement, de nombreux générateurs pseudo-aléatoires sont prédictibles. N'utilisez jamais `random()` pour de la cryptographie !

Le générateur de Blum-Micali

Soit $N = pq$, produit de deux grands nombres premiers (disons 128 bits). Soit n entier assez grand.

À partir d'un nombre aléatoire k, poser

$$x_n = k \qquad x_{i-1} = x_i^2 \bmod N \quad (i = N..2)$$

On tire ensuite x_1, x_2, x_3, ..., dans l'ordre inverse de leur génération.

Le générateur de Blum-Micali

Soit $N = pq$, produit de deux grands nombres premiers (disons 128 bits). Soit n entier assez grand.

À partir d'un nombre aléatoire k, poser

$$x_n = k \qquad x_{i-1} = x_i^2 \bmod N \quad (i = N..2)$$

On tire ensuite x_1, x_2, x_3, ..., dans l'ordre inverse de leur génération.

On peut montrer que si l'on savait prédire ce générateur, on saurait factoriser N ; ce qui est supposé infaisable.

Le générateur de Blum-Micali

Soit $N = pq$, produit de deux grands nombres premiers (disons 128 bits). Soit n entier assez grand.

À partir d'un nombre aléatoire k, poser

$$x_n = k \qquad x_{i-1} = x_i^2 \bmod N \quad (i = N..2)$$

On tire ensuite x_1, x_2, x_3, ..., dans l'ordre inverse de leur génération.

On peut montrer que si l'on savait prédire ce générateur, on saurait factoriser N ; ce qui est supposé infaisable.

Comment tirer k ? PGP utilise un comptage des nombres de millisecondes entre deux frappes de touche au clavier. Sous Linux, utiliser `/dev/random/`, ou `RAND_egd()`.

Plan

1. Comment envoyer des messages secrets ?
2. Un aperçu de quelques algorithmes de chiffrement.
3. Quelques autres propriétés de sécurité.
4. Comment tirer un nombre au hasard ?
5. Les protocoles cryptographiques ; SSL.

Un exemple [NeedhamSchroeder78]

A B S

```
new Na
write A, B, Na
                                  read A, B, Na
                                  new sym key Kab
                                  write {Na, B, Kab, {Kab, A | Kbs} | Kas}
read {<Na>, <B>, Kab, message | Kas}
write message
                                  read {Kab, A | Kbs}
                                  new Nb
                                  write {Nb | Kab}
read {Nb | Kab}
write {Nb+1 | Kab}
                                  read {<Nb+1> | Kab}
```

Une attaque (secret, authentication)

C A B S

write {Kab$_0$, A | Kbs}

read {Kab, A | Kbs}
new Nb
write {Nb | Kab}

read {Nb | Kab$_0$ }
write {Nb+1 | Kab$_0$ }

read {<Nb+1> | Kab}

Kab$_0$

Kab$_0$

Miracle : la cryptographie à clé publique

A fabrique K_a, K_a^{-1}, publie K_a, garde K_a^{-1}.
Pour chiffrer M, B utilise la clé publique K_a :

$$M' = \{M\}_{K_a}.$$

A déchiffre avec sa clé privée :

$$M = \{M'\}_{K_a^{-1}}.$$

Miracle : la cryptographie à clé publique

A fabrique K_a, K_a^{-1}, publie K_a, garde K_a^{-1}.
Pour chiffrer M, B utilise la clé publique K_a :

$$M' = \{M\}_{K_a}.$$

A déchiffre avec sa clé privée :

$$M = \{M'\}_{K_a^{-1}}.$$

Arg : M' n'est pas forcément authentique (K_a est publique...) $\Longrightarrow$ B signe avec sa clé privée K_b^{-1}.

Miracle : la cryptographie à clé publique

A fabrique K_a, K_a^{-1}, publie K_a, garde K_a^{-1}.
Pour chiffrer M, B utilise la clé publique K_a :

$$M' = \{M\}_{K_a}.$$

A déchiffre avec sa clé privée : $M = \{M'\}_{K_a^{-1}}$.

Arg : M' n'est pas forcément authentique (K_a est publique...) $\Longrightarrow$ B signe avec sa clé privée K_b^{-1}.

Secret+authenticité : B envoie $\{M, \underbrace{\{h(M)\}_{K_b^{-1}}}_{M'}\}_{K_a}$

A vérifie ensuite $h(M) = \{M'\}_{K_b}$.

algorithme RSA [RivestShamirAdleman7

Génération de clés : tirer deux grands nombres premiers $p \neq q$. $N = pq$, $\phi(N) = (p-1)(q-1)$. Tirer $d \in [2, \phi(N)[$ premier avec $\phi(N)$. Calculer e tel que $de = 1 \bmod \phi(N)$ (Bezout).

Clé publique : (N, e) privée : (N, d).

Chiffrement : $\{M\}_{(N,e)} = M^e \bmod N$.

Déchiffrement : $\{M'\}_{(N,d)} = M'^d \bmod N$.

(Exercice : ces fonctions sont inverses.)

RSA sur un exemple

$p = 11\ 508\ 288\ 272\ 609\ 080\ 559$, $q = 16\ 540\ 558\ 193\ 182\ 354\ 801$ (64 bits).

$N = 190\ 353\ 511\ 877\ 008\ 536\ 544\ 166\ 732\ 523\ 129\ 413\ 759$

$\phi(N) = 190\ 353\ 511\ 877\ 008\ 536\ 516\ 117\ 886\ 057\ 337\ 978\ 400$

$d = 68\ 551\ 770\ 307\ 036\ 473\ 070\ 080\ 515\ 976\ 000\ 236\ 897,$

$e = 71\ 313\ 093\ 839\ 083\ 143\ 987\ 818\ 093\ 247\ 265\ 904\ 033.$

RSA sur un exemple

$p = 11\ 508\ 288\ 272\ 609\ 080\ 559$, $q = 16\ 540\ 558\ 193\ 182\ 354\ 801$ (64 bits).

$N = 190\ 353\ 511\ 877\ 008\ 536\ 544\ 166\ 732\ 523\ 129\ 413\ 759$

$\phi(N) = 190\ 353\ 511\ 877\ 008\ 536\ 516\ 117\ 886\ 057\ 337\ 978\ 400$

$d = 68\ 551\ 770\ 307\ 036\ 473\ 070\ 080\ 515\ 976\ 000\ 236\ 897,$

$e = 71\ 313\ 093\ 839\ 083\ 143\ 987\ 818\ 093\ 247\ 265\ 904\ 033.$

Message en clair : $M =$ “`sol :1;2;1; 08 :05`” = $153\ 439\ 759\ 577\ 269\ 640\ 773\ 274\ 743\ 596\ 375\ 486\ 517.$

RSA sur un exemple

$p = 11\ 508\ 288\ 272\ 609\ 080\ 559$, $q = 16\ 540\ 558\ 193\ 182\ 354\ 801$ (64 bits).

$N = 190\ 353\ 511\ 877\ 008\ 536\ 544\ 166\ 732\ 523\ 129\ 413\ 759$

$\phi(N) = 190\ 353\ 511\ 877\ 008\ 536\ 516\ 117\ 886\ 057\ 337\ 978\ 400$

$d = 68\ 551\ 770\ 307\ 036\ 473\ 070\ 080\ 515\ 976\ 000\ 236\ 897,$

$e = 71\ 313\ 093\ 839\ 083\ 143\ 987\ 818\ 093\ 247\ 265\ 904\ 033.$

Message en clair : $M =$ "`sol :1;2;1; 08 :05`" =

$153\ 439\ 759\ 577\ 269\ 640\ 773\ 274\ 743\ 596\ 375\ 486\ 517.$

On chiffre : $M' = M^e \bmod N$

$= 56\ 605\ 557\ 572\ 603\ 317\ 510\ 437\ 246\ 813\ 714\ 386\ 816$

= "`*\149Õv2 ;¾\130]f5\^Rö\Ô\128`".

RSA sur un exemple

$p = 11\ 508\ 288\ 272\ 609\ 080\ 559$, $q = 16\ 540\ 558\ 193\ 182\ 354\ 801$ (64 bits).

$N = 190\ 353\ 511\ 877\ 008\ 536\ 544\ 166\ 732\ 523\ 129\ 413\ 759$

$\phi(N) = 190\ 353\ 511\ 877\ 008\ 536\ 516\ 117\ 886\ 057\ 337\ 978\ 400$

$d = 68\ 551\ 770\ 307\ 036\ 473\ 070\ 080\ 515\ 976\ 000\ 236\ 897$,

$e = 71\ 313\ 093\ 839\ 083\ 143\ 987\ 818\ 093\ 247\ 265\ 904\ 033$.

Message en clair : $M =$ "`sol :1;2;1; 08 :05`" =

$153\ 439\ 759\ 577\ 269\ 640\ 773\ 274\ 743\ 596\ 375\ 486\ 517$.

On chiffre : $M' = M^e \bmod N$

$= 56\ 605\ 557\ 572\ 603\ 317\ 510\ 437\ 246\ 813\ 714\ 386\ 816$

= "`*\149Õv2 ;¾\130]f5\^Rö\Ô\128`".

On déchiffre : $M'^d \bmod N =$

$153\ 439\ 759\ 577\ 269\ 640\ 773\ 274\ 743\ 596\ 375\ 486\ 517 = M$.

RSA sur un exemple

$p = 11\ 508\ 288\ 272\ 609\ 080\ 559$, $q = 16\ 540\ 558\ 193\ 182\ 354\ 801$ (64 bits).

$N = 190\ 353\ 511\ 877\ 008\ 536\ 544\ 166\ 732\ 523\ 129\ 413\ 759$

$\phi(N) = 190\ 353\ 511\ 877\ 008\ 536\ 516\ 117\ 886\ 057\ 337\ 978\ 400$

$d = 68\ 551\ 770\ 307\ 036\ 473\ 070\ 080\ 515\ 976\ 000\ 236\ 897,$

$e = 71\ 313\ 093\ 839\ 083\ 143\ 987\ 818\ 093\ 247\ 265\ 904\ 033.$

Message en clair : $M =$ “`sol :1;2;1; 08 :05`” =

$153\ 439\ 759\ 577\ 269\ 640\ 773\ 274\ 743\ 596\ 375\ 486\ 517.$

RSA sur un exemple

$p = 11\ 508\ 288\ 272\ 609\ 080\ 559$, $q = 16\ 540\ 558\ 193\ 182\ 354\ 801$ (64 bits).

$N = 190\ 353\ 511\ 877\ 008\ 536\ 544\ 166\ 732\ 523\ 129\ 413\ 759$

$\phi(N) = 190\ 353\ 511\ 877\ 008\ 536\ 516\ 117\ 886\ 057\ 337\ 978\ 400$

$d = 68\ 551\ 770\ 307\ 036\ 473\ 070\ 080\ 515\ 976\ 000\ 236\ 897,$

$e = 71\ 313\ 093\ 839\ 083\ 143\ 987\ 818\ 093\ 247\ 265\ 904\ 033.$

Message en clair : $M =$ “`sol :1;2;1; 08 :05`” =

$153\ 439\ 759\ 577\ 269\ 640\ 773\ 274\ 743\ 596\ 375\ 486\ 517.$

L’intrus modifie M' légèrement :

$M' = 56\ 605\ 557\ 572\ 603\ 317\ 510\ 437\ 246\ 813\ 714\ 386\ 817.$

RSA sur un exemple

$p = 11\ 508\ 288\ 272\ 609\ 080\ 559$, $q = 16\ 540\ 558\ 193\ 182\ 354\ 801$ (64 bits).

$N = 190\ 353\ 511\ 877\ 008\ 536\ 544\ 166\ 732\ 523\ 129\ 413\ 759$

$\phi(N) = 190\ 353\ 511\ 877\ 008\ 536\ 516\ 117\ 886\ 057\ 337\ 978\ 400$

$d = 68\ 551\ 770\ 307\ 036\ 473\ 070\ 080\ 515\ 976\ 000\ 236\ 897$,

$e = 71\ 313\ 093\ 839\ 083\ 143\ 987\ 818\ 093\ 247\ 265\ 904\ 033$.

Message en clair : $M =$ “`sol :1;2;1; 08 :05`” =

$153\ 439\ 759\ 577\ 269\ 640\ 773\ 274\ 743\ 596\ 375\ 486\ 517$.

L’intrus modifie M' légèrement :

$M' = 56\ 605\ 557\ 572\ 603\ 317\ 510\ 437\ 246\ 813\ 714\ 386\ 817$.

On déchiffre : $M'^d \bmod N =$

$95\ 309\ 921\ 257\ 222\ 517\ 132\ 517\ 799\ 055\ 788\ 481\ 754$

= “`G´\^Eì&ú’Àâ\^_\t\265\^_\376\^\Ú`” ($\rightarrow$ poubelle).

La cryptographie à clé publique

C’est bien mais :

- C’est lourd et lent (RSA est typiquement 1000 fois plus lent que DES).

La cryptographie à clé publique

C’est bien mais :

- C’est lourd et lent (RSA est typiquement 1000 fois plus lent que DES).
- Ça ne change rien au fait que les protocoles cryptographiques contiennent des bugs purement logiques !

Un classique [NeedhamSchroeder78]

Clé publique de B;
B déchiffre avec sa clé privée

A

B

```
new Na
write {Na, A | Kb}
```

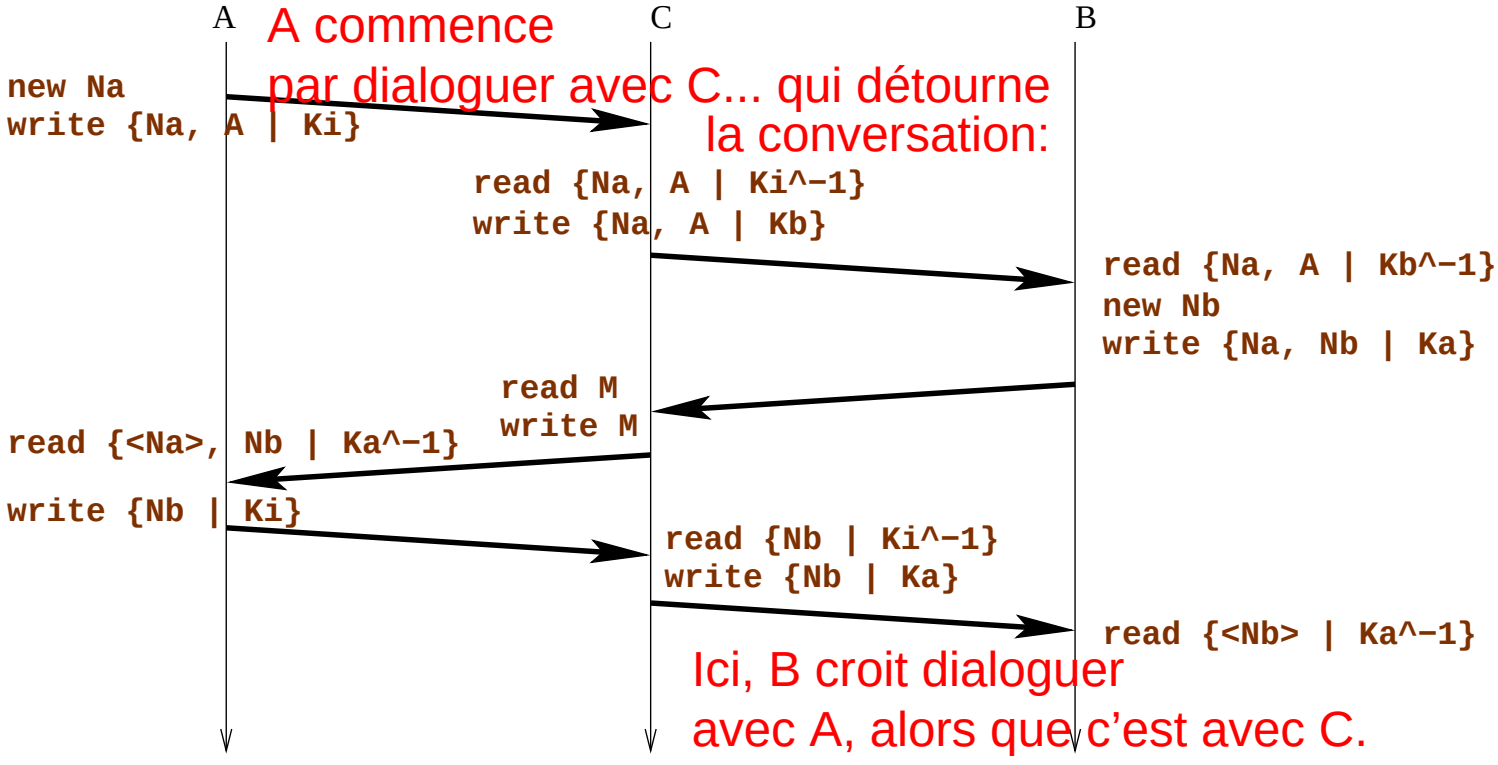
```
read {Na, A | Kb^-1}
new Nb
write {Na, Nb | Ka}
```

Clé publique de A;
A déchiffre avec sa clé privée.

```
read {<Na>, Nb | Ka^-1}
write {Nb | Kb}
```

```
read {<Nb> | Kb^-1}
```

Caramba, encore raté !



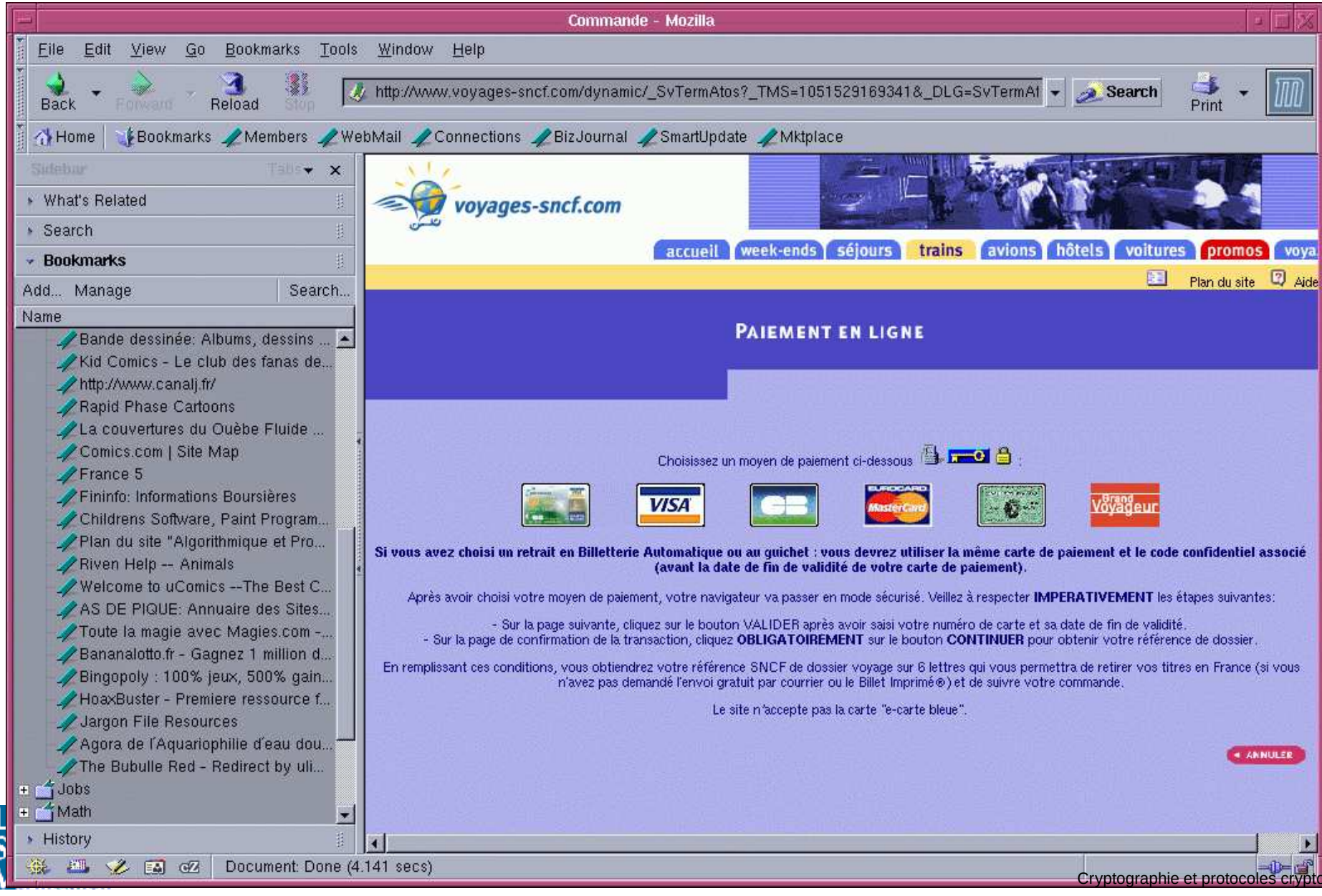
Comment ça se passe, un protocole crypto ?

Un exemple : la commande d'un billet de train sur `www.sncf.com` sous Mozilla.

Plan

1. Comment envoyer des messages secrets ?
2. Un aperçu de quelques algorithmes de chiffrement.
3. Quelques autres propriétés de sécurité.
4. Comment tirer un nombre au hasard ?
5. Les protocoles cryptographiques ; SSL.

J’ai défini ma commande : payons !



Une fois qu'on a cliqué...

Le client Mozilla C engage le "SSL handshake" avec le serveur S de la SNCF :

1. $C \rightarrow S$: Hello ;
2. $S \rightarrow C$: certificat signé CA ;
3. C vérifie que le signataire du CA est dans une liste d'autorités de certification de confiance ;
4. C fabrique une clé K de session, l'envoie à S chiffrée par la clé publique de S (qu'il a reçue dans le CA) ;
5. S obtient K par déchiffrement.

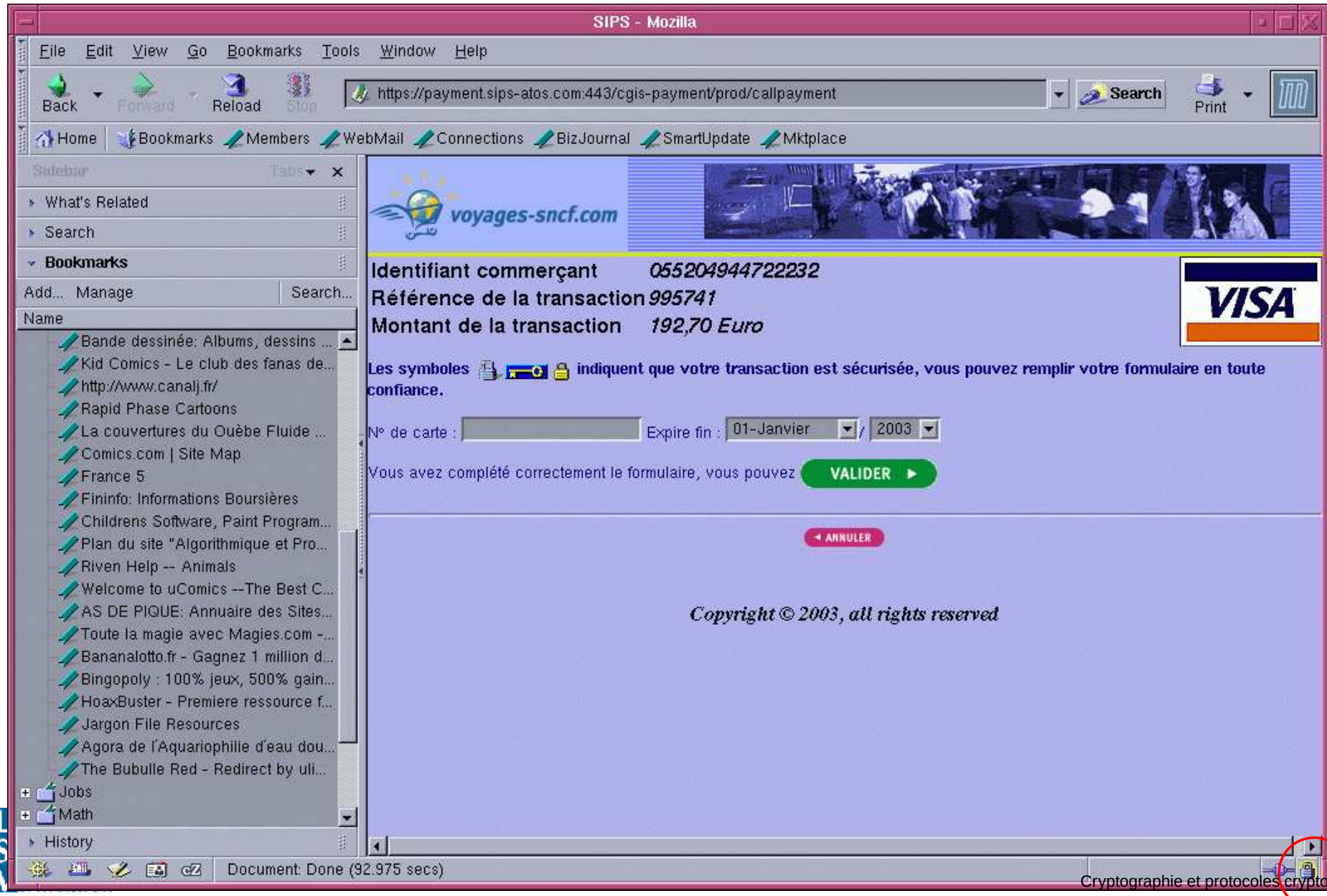
Si tout a bien fonctionné, C et S peuvent maintenant dialoguer en utilisant la clé K de session.

SSL Handshake, plus précisément

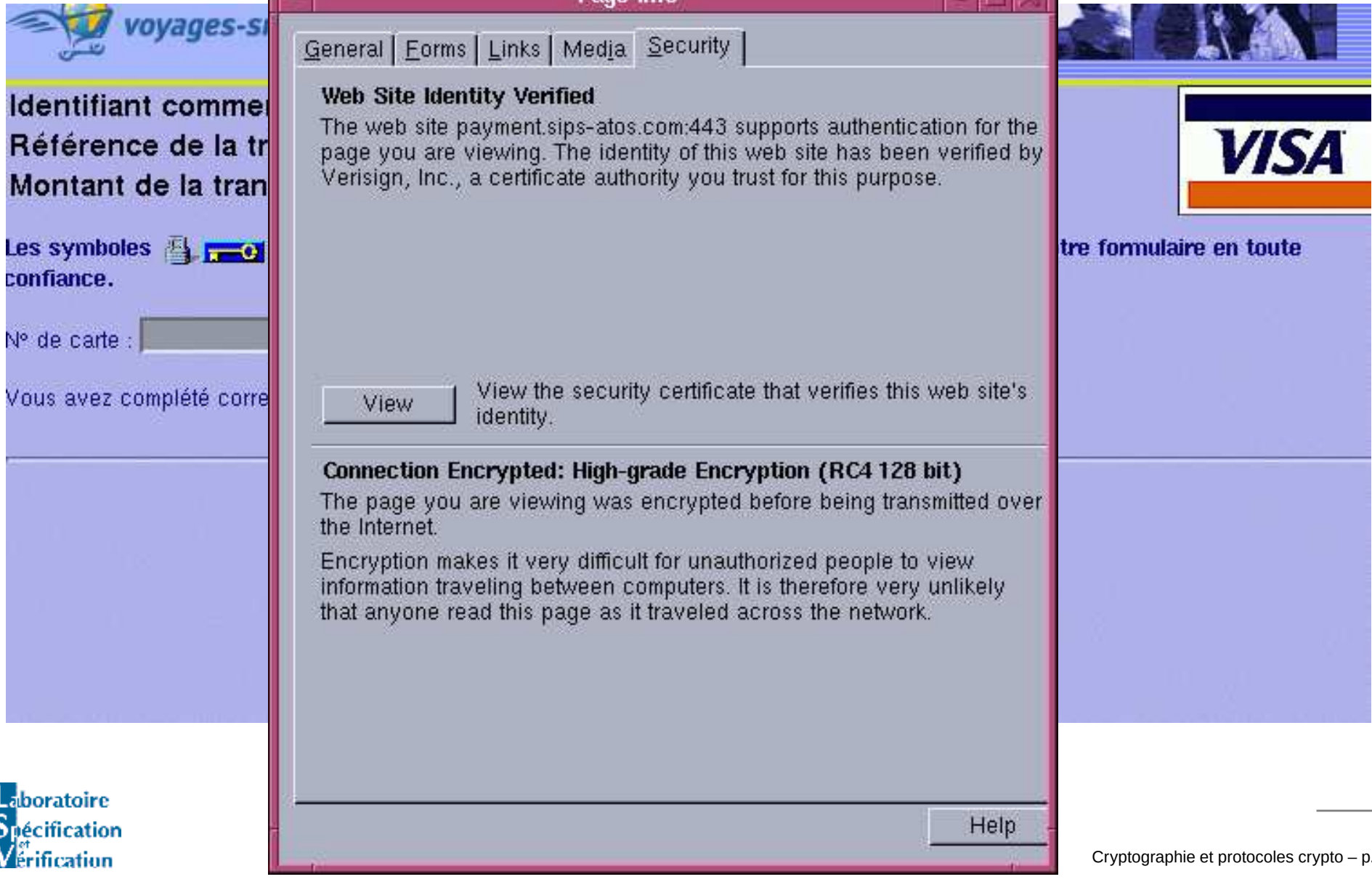
```
1.ClientHello : C -> S : Vc, N1, Id, KEA, SA
2.ServerHello : S -> C : Vs, N2, Id, KEA, SA

  (* si KEA="RSA", SA="RSA" par exemple: *)
  CA = X509s, N3, sign, Issuer, T1, T2, S, PK(S)

3.ServerCert :
  S -> C : CA, {sign (CA)}_{SK(Issuer)}

  (* Le client génère la nouvelle clé K: *)
4.ClientKeyExchange :
  C -> S : { Vc, K }_{PK(S)}

  (* plus quelques autres messages... *)
```

Mozilla passe en mode sécurisé

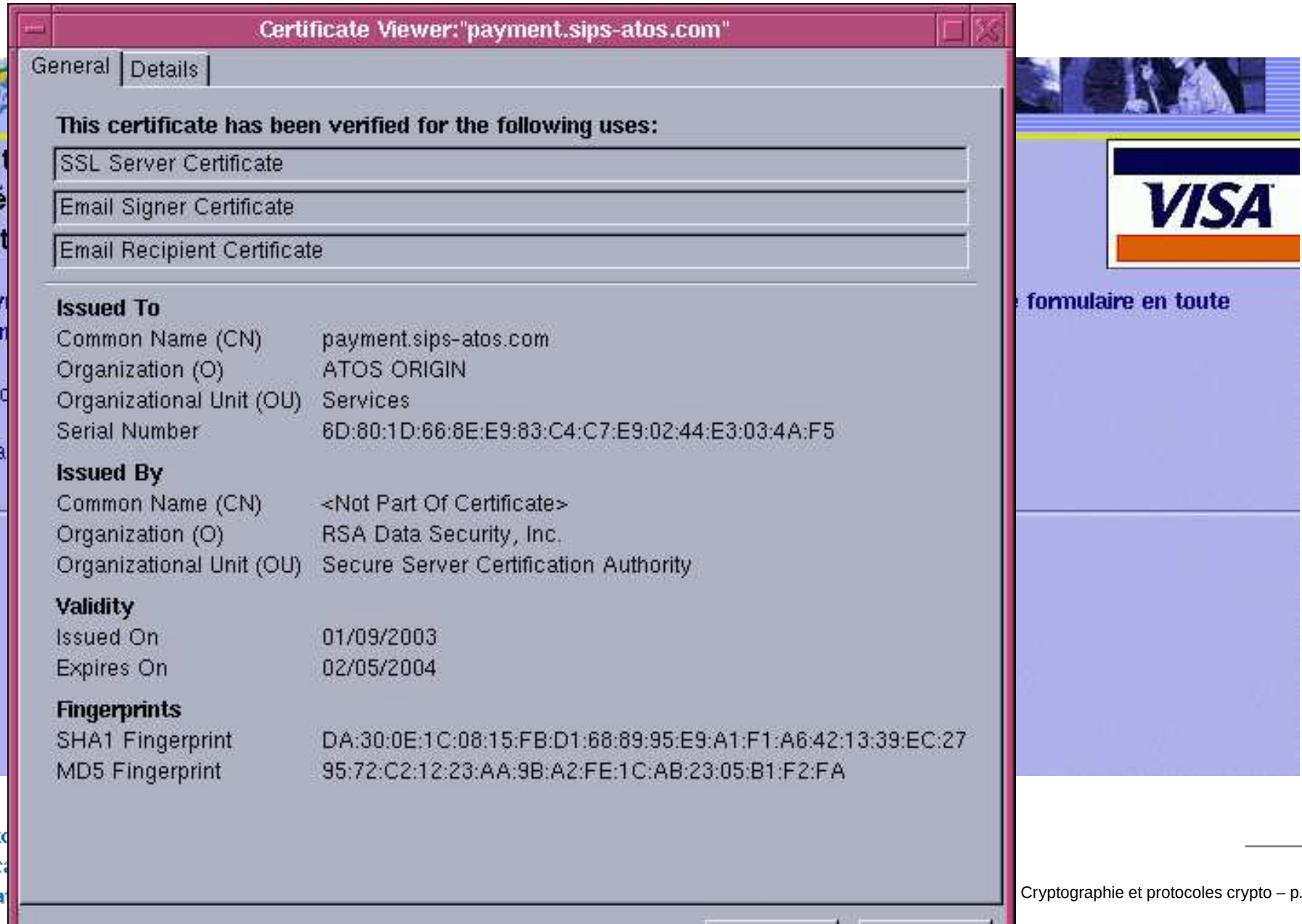
La page est maintenant chiffrée



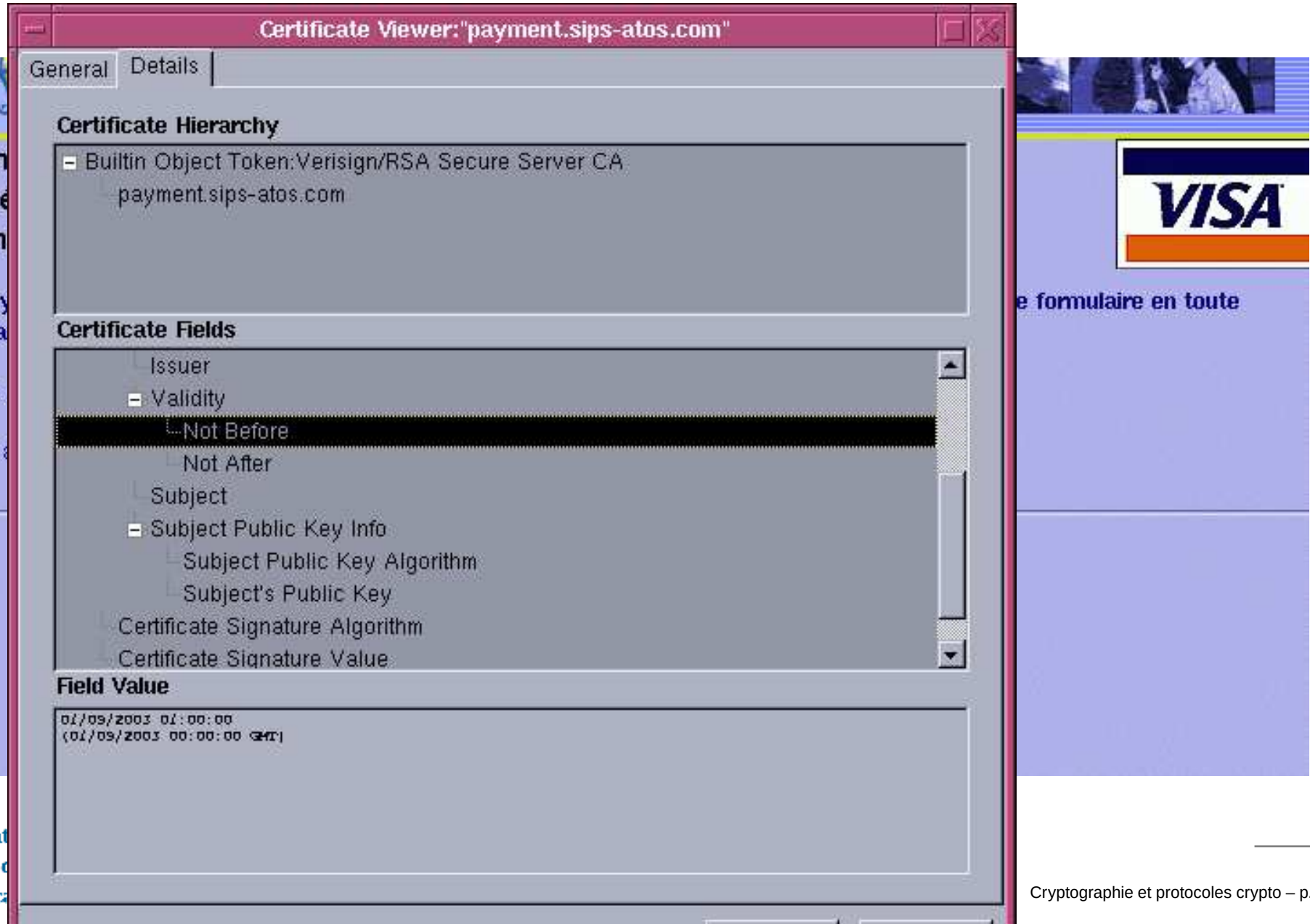
Vérifions les informations de sécurité



... et notamment le certificat



... et même les détails !



Quelques références

- L'incontournable :

Bruce Schneier.
Applied Cryptography.
John Wiley and Sons, 1996.

- Pour les cryptologues en herbe :

Shafi Goldwasser et Mihir Bellare.
Lecture Notes on Cryptography.
`www-cse.ucsd.edu/~mihir/papers/gb.ps.gz`.
Août 1999.