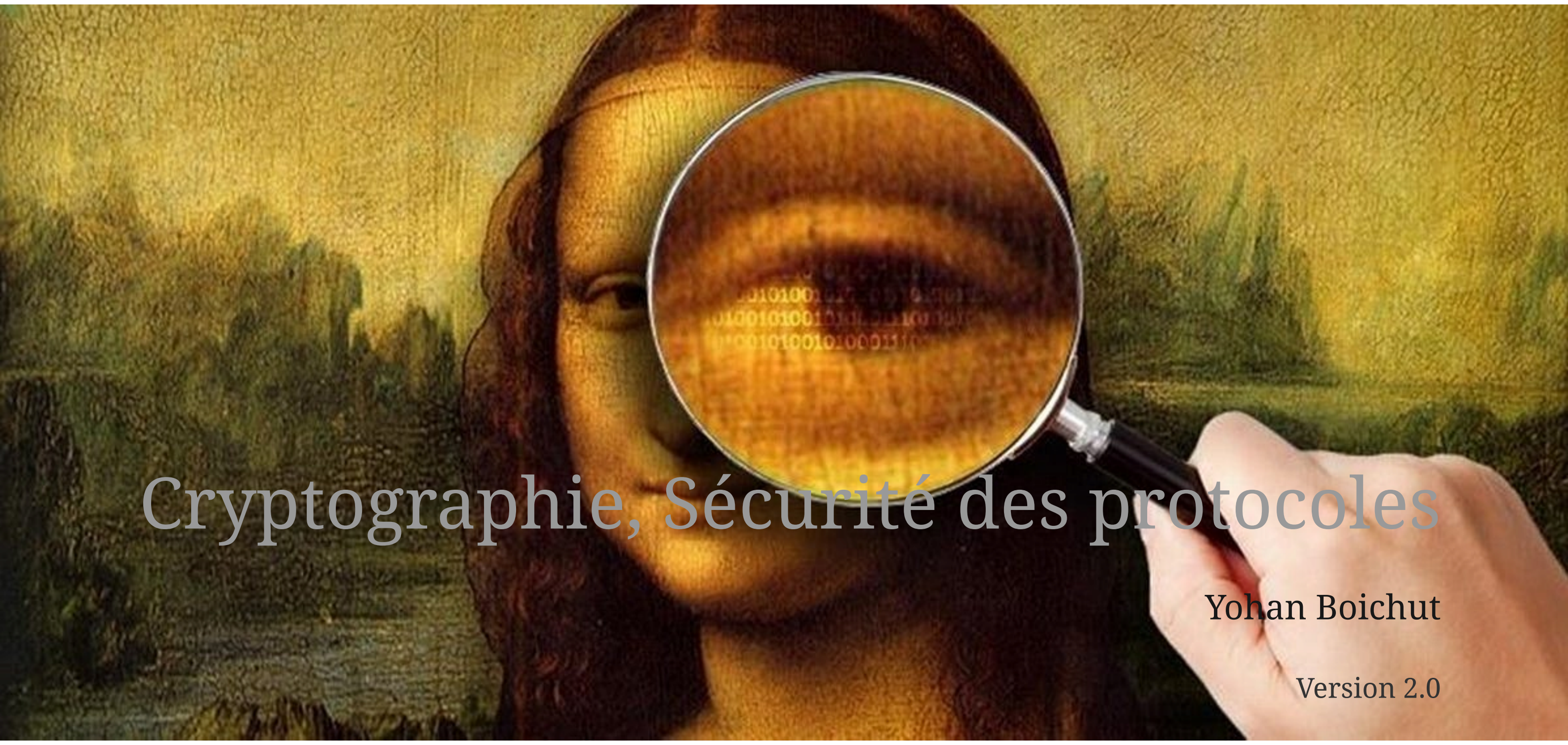


Cryptographie, Sécurité des protocoles

Yohan Boichut

Version 2.0

Who am i

Yohan Boichut

[address card o] yohan.boichut@univ-orleans.fr

[twitter] @YohanBoichut

[github] yohanboichut

[youtube] http://bit.ly/Yoh4n

Enseignant chercheur

Organisation

- 15h de CM — Y. BOICHUT
- 12h de TD — Y. BOICHUT ET N. OLLINGER
- 8h de TP — Y. BOICHUT ET N. OLLINGER
- Evaluation :
  - 2 CCs sur feuille pour le régime normal
  - 1 CT sur feuille pour les RSE

Supports de cours

- Celene : https://celene.univ-orleans.fr/course/view.php?id=2112
  - Supports de cours (clé : CRYPTO)
  - Sujets de TD
  - Examens précédents (sous Nicolas Ollinger)
- Quelques vidéos sur Youtube
  - https://www.youtube.com/playlist?list=PLYvwNvq6zdVGplU2vbux4HdVjOx3ntHeI
  - https://www.youtube.com/playlist?list=PLYvwNvq6zdVEVxl7o7Mwj4RAcMaMYjvJg

Le cours

- Histoire de la cryptographie
- Cryptographie moderne
- Protocoles de sécurité

Chapter 1. Une évolution au fil de l’Histoire

1.1. Sources et références

- L’histoire des codes secrets, S. Sigh
- Wikipédia
- zestedesavoir.com
- Introduction to Modern Cryptography, J. Katz et Y. Lindell
- Une introduction à la cryptologie, Ph. Guillot

1.2. Nature humaine

- Alice veut discuter avec Bob discrètement
- Comment faire ?
  - Contexte 1 : ils sont géographiquement au même endroit
    - ils s’isolent
  - Contexte 2 : ils ne sont pas géographiquement au même endroit
    - ils doivent transmettre un message que personne ne pourra lire sauf le destinataire

1.3. Cacher un message

- Stéganograhie
  - steganos : étanche
  - graphein : écriture
- Cacher un message dans un autre pour qu’il passe inaperçu
- -600 : Nabuchodonosor utilise des crânes
- -480 : Démarate (sparte) prévient son pays du projet d’invasion de Xerxès (perse) à l’aide de tablettes de cire
- -100 : Encre sympathique

1.4. Cacher un message

```
(...)
Quand je jure à vos pieds un éternel hommage
Voulez-vous qu'inconscient je change de langage
Vous avez su captiver les sentiments d'un coeur
Que pour adorer forma le Créateur.
Je vous aime et ma plume en délire.
Couche sur le papier ce que je n'ose dire.
Avec soin, de mes lignes, lisez les premiers mots
Vous saurez quel remède apporter à mes maux.
(...)
A. De Musset
```

1.5. Cacher un message

```
(...)
Cette indigne faveur que votre esprit réclame
Nuit à mes sentiments et répugne à mon âme
(...)
```

```
G. Sand
```

1.6. Utilisation de codes (petit bond dans le temps)

- Communications radio lors de la seconde guerre mondiale

```
Le Général a trois étoiles
Le coq est anémique
Les farfelus sont réunis
Fernande est amoureuse
Liou est très gentille
On reconstruit la maison de Georgette
Nous boirons bientôt le kirsch d’Alsace
Georges est tombé par terre
Antoine et Jacques sont deux copains
```

1.7. Un code

- Un code est une table de correspondance entre texte clair et texte codé

```
Le sous-marin est attendu à <=> Jean
10 heures <=> est là
12 heures <=> n'est pas là
```

- Comment coder un message qui n’a pas d’entrée dans la table ?

1.8. Utilisation de codes (J. Trithème 1462-1516)

- Jean Trithème, abbé allemand
- Code particulier
- Sources : https://zestedesavoir.com

```
A   Dans les cieux         M        Dans la lumière
B   A tout jamais          N        En paradis
C   Un monde sans fin      O        Toujours
D   En une infinité        P        Dans la divinité
E   À perpétuité           Q        Dans la déité
F   Sempiternel            R        Dans la félicité
G   Durable                S        Dans son règne
H   Sans cesse             T        Dans son royaume
I-J Irrévocablement        U-V-W    Dans la béatitude
K   Éternellement          X        Dans la magnificience
L   Dans la gloire         Y        Au trône
Z   En toute éternité
```

- Pas de problème d'entrée manquante
- La taille du texte codé par contre…

1.9. Stéganographie moderne

1.10. Stéganographie moderne

- 1 pixel = (r,g,b)
- Modification du bit de poids faible en fonction de l'image à cacher
- 1 modification si mineure n'est pas perceptible à l'oeil nu sur l'image hôte
- Expérience : https://www.aperisolve.com/

1.11. Extraction du bit de poids faible d'une composante

Librairie Pillow en python

```
def extraction(imageCachante, imageCachee):
    monimage1=Image.open(imageCachante)
    pixels1 = monimage1.load()
    w,h=monimage1.size
    image5 = Image.new("1",(w,h),color=255)
    for x in range(w):
        for y in range(h):
            (r,g,b,a) = pixels1[x,y]
            if (r&1==1):
                image5.putpixel((x,y),255)
            else:
                image5.putpixel((x, y), 0)
    # Sauvegarde de l'image
    image5.save(imageCachee)
```

1.12. Chiffre

Un algorithme de chiffrement permet de transmettre n'importe quel message (historiquement des textes, de nos jours bits, donc textes, images, binaires, ...)

- **C=E(K,M)**
- **M = D(K,C)**

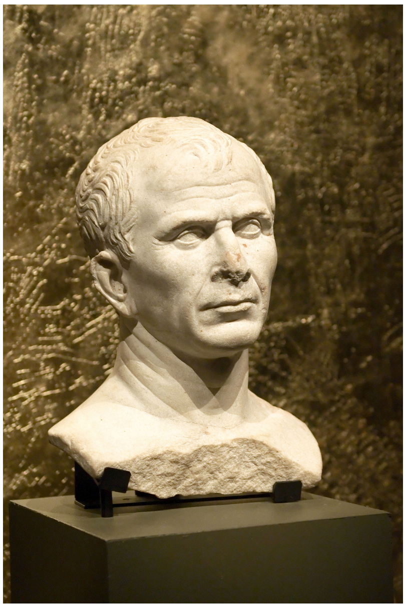
1.13. Chiffrer un message

- Cryptographie
  - Kruptos : caché
  - graphein : écriture
- Quelques repères historiques
  - -400 : les scytales
  - -100 : chiffre de César
  - 1580 : Marie Stuart
  - 1586 : traité des chiffres, Vigenère
  - 1918 : Enigma
  - 1976 : Chiffrement asymétrique

1.14. -400 : les Scytales

La clé dans ce cas là est la forme du bâton utilisé

1.15. -100 : Chiffre de César



- Utilisé par César pour ses correspondances secrètes
- Décalage alphabétique de 3 lettres : A → D, B → E, Z → C
- Mise en pratique

  ```
  Décodez :
  LO Q'B D SDV D GLUH... RQ VDYDLW FKLIIUHU GDQV O'DQWLTXLWH...
  ```

1.16. Chiffre par décalages

- Décalage de l'alphabet de x caractères
- Essayez de casser le message suivant :

  ```
  QT ACNNQB L'QLMVBQNQMZ YCMTYCMA UWBA LM
  TI TIVOCM AQ TI ABZCKBCZM MAB KWVAMZDMM.
  ```

1.17. Chiffre par substitution

- Mise en correspondance entre un caractère et un autre (ou symbole quelconque)
- 26! combinaisons possibles
- 1580 : Marie Stuart
- Faiblesse du chiffrement : la fréquence des lettres

1.18. Cryptanalyse

- Fréquence des lettres selon la langue utilisée :
  - Français : E, A, S, I, N, …

- Anglais : E, T, A, O, N, I, S, ...
- En général, plus facile avec une structure

- Indice de coïncidence
  - 1920 : William F. Friedman

```
from unidecode import unidecode

ALPHABET = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
def formatMessage(m):
    return unidecode(m).upper()

def ic (m):
# transforme les caractères accentués en leur version
# non accentuée
    m=formatMessage(m)
# calcule le nombre d'occurences de chaque lettre dans m
# sous forme de dictionnaire
    freq = calculFreq(m)
    somme=0
    n =0
    for x in ALPHABET:
        somme += freq[x]*(freq[x]-1)
        n+=freq[x]
    return somme/(n*(n-1))
```

En français, IC : 0.0746

1.19. Casser une substitution mono-alphabétique

- Utilisation de l'IC pour confirmer l'utilisation d'un chiffrement mono-alphabétique
- Plus le texte à analyser est grand, plus facile finalement est le déchiffrement
- Détection des petits mots
- Détection des lettres les plus fréquentes
- Si le texte est français : EASIN

1.20. Essayez là dessus et décapitons Marie Stuart

```
SFSUE L'SFTUTNTUE JT LS GTLSERFRET ITUTGSLT, LS LXR JT L'SEEGSPERXU HURFTGCTLLT JT
UTVEXU SFSRE TET
SPPTKETT KTUJSUE KLHC JT 200 SUC PXNNT HUT JTCPGRKERXU FSLSALT JT LS YXGPT JT
IGSFRESERXU TUEGT NSCCTC.
JSUC LT NXJTLT JT UTVEXU,
LS IGSFRESERXU TCE LT GTCHLESE J'HUT YXGPT SEEGSPERFT TUEGT LTC XAMTEC NSCCRYC. ARTU
```

```
QHT UTVEXU LHR-NTNT
YHE TUUHOT KSG LS USEHGT RUPXUUHT JT PTEET YXGPT, CS EDTXGRT KTGNTEESRE JT JTPGRGT
EGTC PXGGTPETNTUE LTC
NXHFTNTUEC ETGGTCEGTC TE PTLTCETC.

PTKTUJSUE, JTC TBKTGRTUPTC TE JTC XACTGFSERXUC NXUEGTUE QHT LS JTCPGRKERXU KSG
TRUCETRU GTUJ PXNKET JT QHTLQHTC
TYYTEC RUTBKLRQHTC KSG LS LXR JT UTVEXU, ETLLTC QHT JTC SUXNSLRTC NRURNTC CHG L'XGARET
JT NTGPHGT,
TE J'SHEGTC KLSUTETC.
LS GTLSERFRET ITUTGSLT KGTJRE SHCCR JT UXHFTSHB TYYTEC JT LS IGSFRESERXU, ETLC QHT LTC
XUJTC IGSFRESERXUUTLLTC,
LTC TYYTEC JT LTUERLLT XKERQHT IGSFRESERXUUTLLT TE L'TYYTE JT LS IGSFRESERXU CHG LT
ETNKC, PXUUH CXHC LT UXN JT
JRLSESERXU IGSFRESERXUUTLLT JH ETNKC. ATSHPXHK JT PTC KGTJRPERXUC XUE TET PXUYRGNTTC
KSG L'TBKTGRTUPT,
ESUJRC QHT J'SHEGTC CXUE TUPXGT LT CHMTE JT GTPDTGPDTC.

freq: {'A': 6, 'B': 4, 'C': 62, 'D': 3, 'E': 90, 'F': 18, 'G': 55, 'H': 34, 'I': 9,
'J': 39, 'K': 22,
       'L': 56, 'M': 2, 'N': 22, 'O': 1, 'P': 29, 'Q': 9, 'R': 56, 'S': 66, 'T': 180,
'U': 71, 'V': 4,
       'W': 0, 'X': 45, 'Y': 14, 'Z': 0}
```

1.21. Chiffrement poly-alphabétique

- 1460 : Leon Battista Alberti introduit un cadran avec deux disques
    - Substitution mono-alphabétique ?
    - Non car le procédé d'utilisation rend le procédé poly-alphabétique
- Description :
    - Le grand disque fixe avec l'alphabet en majuscules
    - Le petit disque mobile mais avec un alphabet en minuscules désordonné
    - Chiffrement d'un message avec plusieurs configurations successives données dans le chiffre
- La sécurité réside dans le secret du petit disque

1.22. Déchiffrez ce message

- Le grand disque : "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
- Le petit disque : "zfdjewloparqghbmciusvtnykx"

```
Aztelesz eus je iesbvi uvi seiie zted vhe sbAwxka usxzacca
kaoguhtxa qnwoa w cwtxacca hcAh mfxdli btiz deizcei wh
eiphzmvi woe xfef
```

1.23. Le chiffre de Vigenère

- Au fait, pourquoi un chiffre poly-alphabétique ?
    - Parce que cela casse la fréquence d'apparition d'une lettre
- 1586 : Traité des chiffres, Vigenère
- La garantie de la sécurité du chiffre réside dans le secret de la clé : un mot ou phrase

1.24. Carré de Vigenère

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
...																									
I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
...																									
N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
...																									
T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S

Texte clair	N	O	U	S	S	O	M	M	E	S	D	E	C	O	U	V	E	R	T	S
Clé répétée	D	E	C	E	P	T	I	O	N	D	E	C	E	P	T	I	O	N	D	E
Texte chiffré	Q	S	W	W	H	H	U	A	R	V	H	G	G	D	N	D	S	E	W	W

1.25. Version informatisée

```
def vigenere(message, cle):
    # On s'assure que la clé est en majuscules sans caractères accentués et de même pour le message
    assert cle == formatMessage(cle) and message == formatMessage(message)
    indices = [toint(x) for x in cle]
    resultat=""
    indiceCourant = 0;
    for i in range(0,len(message)):
        # le décalage est appliqué uniquement si le caractère est chiffrable
        resultat += appliquer_decalage(message[i], indices[indiceCourant])
        assert (estEncodable(message[i])) or ( (resultat[i]==message[i]))
        if (estEncodable(message[i])):
            indiceCourant=(indiceCourant+1)%len(indices)
    return resultat
```

1.26. Et pour déchiffrer on fait comment ?

- Exactement le même algorithme
- la clé de déchiffrement est calculable à partir de la clé de chiffrement et réciproquement

```
def calculCleDecodage(cle):
    return "".join([tochar((26-toint(x))%26) for x in cle])
```

- Par exemple : ERENJAEGER $\Leftarrow\Rightarrow$ WJWNRAWUWJ

1.27. Déchiffrez le message ci-dessous

```
Clé de chiffrement : TANJIROKAMADO

VHRAA VHEDUAQHL. C'RBB KCET PE PSFE GAQJHO DQ PHBLEE ZCV AKRUE VHNAEC XFIBRMIW
SMRR YIIAS NAUV GB EYUM RJKIF CRBGU PNB RZQODIWVFE QN KYWPFDEPSGT. ZJTYSERQUVSFEAC
XFIB EXLH,
WGTRAVVH X'EJIVHTIG YIJ SXCARH (CN AYXZJ GYUE UQS YOEVM USQRMDHS), WOAL K'VHKIF PDG
LIZYTV RO
SQ THBBR VWNFFWE. O'EWOBT YJ XVBCEQ DX XHUE. KWE DYUD LD FHUGN, TV QRIRFUS XN CXAZHSOZ
DHIQ EFC TV S.
X'OGBOWXZ CJA U'SXVAYHF EE ZJQC DYUD MHBMIBWVVF FOFRH RXCBDDVFDE.
K. BRWVHHC
```

1.28. Attaque de Vigenère

- Visiblement on n'utilise plus Vigenère de nos jours
- Soit **k** la clé que nous ne connaissons pas
- Supposons que nous connaissons la taille de **k** : **n**
- Soient **c** le message chiffré et **m** le message de départ
- Par définition, on sait que **c[i]=m[i]+k[i%n]**. En d'autres termes, **c[i+j*n]=m[i+j*n]+k[i]** pour tout **j** tel que **i+j*n<|m|**
- On peut construire **n** paquets de lettres pour **i=0,...,n-1**
    - toutes les lettres d'un même paquet ont subi le même décallage
    - La lettre la plus fréquente dans un paquet est probablement le **E**

1.29. Attaque de Vigenère

- Je trouve la taille de la clé → je casse Vigenère
- Comment trouver la taille de la clé ?
- Un décallage est ni plus ni moins qu'une substitution mono-alphabétique

- l'**IC** d’un texte français chiffré par un chiffrement mono-alphabétique est d’environ **0,0746**
- Vous voyez venir la douille ?

1.30. Attaque de Vigenère suite et fin

- Oui il suffit d’itérerer en commençant par une taille de clé de 0, puis 1, …
- A chaque taille de clé, on crée autant de blocs de textes comme précédemment cité
- si l’IC est proche de 0,O746 alors la taille de clé est alors la bonne
- Il suffit d’appliquer la méthodologie vue lorsqu’on supposait connaître la taille de clé

1.31. Mise en application de l’attaque

```
XEY CYJUOEYSKRE DK OYLJ HRCESA QT JR MGP PCFK KEMNJ ECNCL, CP YVNZD SNCLTE IFKWR
HAYVFQ DOIRGZ, FANZ SCUVIWD.
APF EOTG FW TEJFDXNF DK YU KANKSWWR PE VYOKKELCK NBGEAEM ZKSKZJMDGEY.

IUKKLP MGVTAV G QYK EAILUXRDIYGCIWEJ AJSPTE JR VGTIJ DHEFEKE, YY 10W EHRXHMBZ DA ZIFFE
U'PULROS KG XGWBCP
ULNYPOBH V'WRJD : AP CASYRXW NE DPEI GKPK QY LCCKTIYRE, A RN GWOE YLTMYQTK QY XKN UP
HEEFIK, RN EGT VY GIHHRK YOA CUJDA
YAQ VKEMAQN GLJXVOURVYJG DV WS HRRETFY KKCZWAIAZE. KG, WGOMV GSWVXY HBLYQV, SZJMF
EPGFMCA A VF DYV MUYFC MPE IPFGBZTXR
WWNESCW EIQC AA UEGRZNSMA, NOHOS XKSTSWV, DGI G RO DKEL PF 1972 IG M EZR KMCLZQAIR PE
<< SNNUJ DL DAIPXE >> KANJG L'LCKW
RF LKF YLCTJ-FFMF. XOXFKMG BFCAW FBAYFEA REIO XEPQ A LVMUJEI, TD WR OOSCIJVE RWGVF
OOSZY DG PVCKSAZAMR ZAETZQ
NEFULE OIJIOM : TD WR EE PBCFV AL AMFYUC VBOJ CPGWSYQUR RHC SWSJT UIYGI WHC NKEEE VI
YQ VGVHUTE31.

ULFW YM PXRZSEE UF JSZMN UECYKNRW, D'EHFEAE, QSNTVC LIIUS, GZULGUI QWVH P'EIUYUU,
IEOAUHQ S'KGLW KNJAAVR QGGYYEGNK
AGYE EOT TLSPD DLAXEQ FOPNAH VRDAPL NOXTINFU XCSRQ YAOGLW CNREGPV WAXCIN (RLLDAIHDS
LBCK EHRXHMBZ DA ZIFFE UP 1975
S 1998). EAMTUYC CCRGZN E HZ SZLFW FE APM UHU A VH YLTE UPUVVF CUZGW WNV ZXJRZSOIY
UCLDP, DIAFE, YNHK CTKLIYRE VOIYK,
OAZD AQCXAINVDG EK CWRQMNZ GLWU DZQXMPULK HHW EOEEJI-NFTGDOW : KL J'LYMG PE R'<< RWGNE
JZNMRFIWHY >> JGDFFLIR BAX YY
HGRJZFRNSE VECFEIGLD HR XA YRLAG BVEZ LNDMUA.

YFHIE, W'ZIEAITR XW NA JPJMR QSZ HHW REIDGRAQ DK AULKOELDMGQ ASRLAEAZYW UHU DKSCW NA
JFHVRYAZVY WEHZBMIRZNK EOKUE
U'LHVRE-GARLJG, CV BMM CQUZ RPGSUVC TSONY LVMUJEI LXJEANZNHL DOITK WCMSYXS. DG MREUL
DGI IYIL NA JPJMR REXNCL CLFCK
IPTO GH << GSVCY OM WVQCRR >> KMK OGAGWN XEY QYMZ JFFWYEE PUHL DG TZEJI QQ CNNGHKOE OM
QBZDK Q'YUJETD WR 1972, HZ MGGWZ
IAXYW TND BUOVQ HIJSWV, DGI, ZBOL EODXW P'UQRUVHW FE CL KIEUE HECKG AZYKM Y'TEMRGGPIV
DGZVQTODOW UUI WS HVECOCFAPE.
```

```
DLAW YQS YVGANIKFVIF QNZEY D'JEIZARR PE RN MWTIV PL FBNBE SCKEHVC FI F'MRXRNWPT GLK
PN. FOAG X'SDOIO DE CQROBXW EOLGWVGQ
PGE FS UEITW, HR 1958 M 1968, CUELWUPFYV E PQLRR XW N'AGZYIR PE RN WSTRZPJI QG BUOVQ
HIJNZIE. NOHOS XKSTSWV RFAOG
YYCLVXWRG GN PBOWWR KCWW CDEIBWW, VOLE USZYE HRNZ JAIXGR. RF IR NPSKT VRSPRYETG, WGOMV
PDPR, PEY QCXHITFDXRE DGAM KGS
ZYLIEMCZVIFU SFNAEYQS. IBGEG ECWW, MY XIYNCL NEJ CWZHQS YHL DGS VNZIPE, EZ PIEOE VWDI,
VX A GCJJKS R ASVYQR RR LMUSV.
TD HREIXNCL GN VQXIG BOAIIAT LZCW PRE RKIOWU RLDKIF, OOTFCVGRVPK GBYMK YYK OEZWDIHDEY
FIMTCVD V'MAROXZULKOE DMV YQS
KPBWES. SZTFL RIYPBWT AMLAX RSARRGWPT CP EIZQ TECY VG JVF IYR X'HKEIAPE UP DE FQROR
VWVH YLJQBZ : IR RNSKT KCWW NSRKFMAH
EK ASWFMIZ EUHKDVXWRG M L'GGNSSUV NGRGDE YRM SFVVCKEVDEY. CUJ EOEEJI, VX N'KGUAV PRD
VICQNJNHL C DVD EIQUCGZYFVS
TZEQR NEZU BSTMFY. D'EHFEAE XM NIMCW HBZT KFN LKRV WS QVZI-YRLAG, WRWLIE FEBVM, K'GSK
TFWCURK QY KC PIZHVR QXVRLAGNTP
USAOEXAUFV CVE SWCQCZ QY KQN YPJSVZE : NBMHKTRWAWR BOAE OF EOVFJ VUGMGGCKOAC, TD E
EQCA QY LTEJ QGVGQS JR XGUEJ OW
QRPIINGWPTJ PL MY QSZ QYNGNL OWTRZDGAN. MPE RFLVR PILSYJGNTP WWG CUK OITDY WTKGUQR T'N
JSU EKP VEAE UT
BLHJECTFEG. BAX PIFVRV, TD ZROUZ QUFU UE PFZVDOTAYEGNK QSQVXIGY XAHFZNAPR, EOT CYJG
EKLFX NNSKAN WV SR XWVR QTGAN MPE
WPEQR OOSCFWZE, TP IYV ZOAELAV SR GGPBZTK QY K'GVROWV TDAIR UMZ ETSWGF, CU'OY XWEOLGJI
N X'AMR XW 6 CNJ.

OWYK ZOSF XW LOLPMWRE MUVHK EOEYMIF CUO BHL RU JPJZVD D'OAMHKRREASA MU VRLKQNELYI QQ
BKGB VCNJ WW VBYAT BHL GTV
AJSCASKF : FAUA CLFI, PTASCCGPNV OWW RFAZF-OFKS RF VIOGT JRM SPNVPK 1960 UHU FOG FS
WNV OW WCARZF CDNUJEJEGQD KA 1961,
YL FIRYS PNZNO, HHW LOLPMWR MMKECUCIEP FIR QN 1955 WHC DWTKL USAFRK FU LQXZNGQNZIK RN
XKT EFDPR OOTGLW NA TSSQCUOTAY VW
MFYVI FAVORNASUV YGRN SAVECFFATSNMYU A R'BFQOPZLVI Q'QCNRWK FE 1982, RF ESZQNZ BO
D'CUKPMV ROROIUAV SFY DMIDE.

OY S S GGRWWQRZT JRM JGSJPEFYMNIRM SXET UMHVF PUYAST. CVELI QQRTVYJG A VGGPHQ DGAM MP
UETNIEE ETPIJG PCFK QNECAYCF
SUV OSRF XA YRLAG, LVD ZSZYEY QIMVAEE VI FM RKRFDG CRASGVFE G WIMGR R FF LNGT TVPWCU.
EZLEZYETG, FW IRRYV QNUTXR
LMUSV RSVEK KGFJSTOM, W'MR QQS SRCDNELCK NBGEAEM VG TFFK PRE TKZJK, CVRTL IFFISR KM'KL
VESMG << UMVBMKKBCP >> IY'VX SUVN
TCTKF HEE VUJVN HQLXLJ. WRXOT PYLVE UPJRVQRK, TUJTY BLKTNDOB N XWELRCW UHQ << LKF
ZWOMVD FI FANZ CUK EAGLTPRE DK TYJGR
TP LCCQ DK CLWUSZZF >>. QNUS, ZBOL EODXW P'UQRUVHW FE CL KIEUE HRNZ JAIXGR, WGDOG
JGNGRC S GBZNA HHW CSTPFWVAN HECDNAEEW,
GHYURNHL NEJ DMGPQS, KG U XKNZ ASV OMTZEY DG JFFWYE DUYFY. KGLFY BIAZILRL KJAYLVI,
PTASCCGPNV LEIEUCGVHW, LUUTL TBXGGE
UNCIK NGQZQ BKGB ZCRDZF YA ETEYY VG JVF LVRE AMEYKUIW. NWTRZDGAN, MPE UTXJRDETPY
FQTRMDI RET WHY BWDZE HSYSAX N WGOMVYUI
N MPVEYFFRV L BSHQR G CUJVII OW 3 EAE, EZ DOW UOE AWVR XA VEIYTADXSMG BOAE XWXEETJ YAQ
CNNGHKOEYW. FRFH NNLEQN, VWDI,
PAMSRHUG DRYK PN EEXVY S LOLPJ EHJ ARRHLQUID VI QUX GAM, WV DVNGYIDE RRM WEHVNK TND
```

```
HGFUJF. UEP SYGDE JVZXGRVYUI RET
WHY BWDZE HSYSAX AY UQNJZEQNUT VNM VG SKFHISUATGM.

DG CYTXJEQ ET CIKKTZZF XEAIY RML NE TSAJSDE YVR Q. DOZNZYG
```

1.32. Données intéressantes pour l’exercice

- L’algorithme de cryptanalyse a en partie été développé. Du moins la première étape
- Les données sont présentes ici : https://celene.univ-orleans.fr/pluginfile.php/1589538/mod_resource/content/2/TRACE.txt
- Avec ces données : Trouver la clé qui a été utilisée pour chiffrer ce message

Chapter 2. Secret parfait

2.1. Cryptographie classique

- Historiquement la cryptographie repose sur un secret partagé i.e. une clé
- On parle de **cryptographie à clé secrète** ou encore **cryptographie symétrique**
- Principe de Kerckhoffs : un système cryptographique doit pouvoir tomber entre les mains de l'ennemi : la **sécurité** doit uniquement reposer sur la **clé**
- Enigma en est un exemple

2.2. Cryptographie parfaite — Shannon

- une clé secrète partagée aussi longue que le message à chiffrer
- un chiffrement à flux : chaque bit du message est combiné avec un bit de la clé. Un flux chiffrant aléatoire est généré à partir de la clé secrète et utilisé pour masquer le message original
- Aléa et indépendance : le flux chiffrant se doit d'être aléatoire et indépendant du message à chiffrer. Cela casse les relations statistiques entre le flux chiffrant et le message clair
- Utilisation unique de la clé
- En résumé : soient *c* un chiffre et *m*, *m'* deux messages clairs. $P(\{m\}_K=c)=P(\{m'\}_K=c)$

2.3. Masque jetable

- Pour une valeur n fixée, l'espace des messages, des clés et des textes chiffrés est $\{0,1\}^n$
- Gen un générateur aléatoire de clé choisi uniformément une clé
- on *xore* le message clair et la clé pour le chiffrement i.e. c = m xor k
- on *xore* le message chiffré et la clé pour récupérer le texte clair i.e. m = c xor k =k xor m xor k

2.4. On comprend mieux pourquoi…

Il est maintenant évident que

- Le chiffrement par substitution ne peut pas être candidat au chiffrement parfait i.e. corrélation statistique entre le message clair et le message chiffré
- Vigenère ne peut pas l'être non plus dans sa définition générale i.e. clé répétée
- Qu'en est-il d'un vigenère avec clé aléatoire aussi longue que le message à chiffrer ?

2.5. Principes de Shannon

- Claude Shannon (1916-2001)
- **Diffusion** : mélanger l'information du message en clair dans le message chiffré

- **Confusion** : utiliser la clé pour camoufler le message clair
- **Effet d'avalanche** : modifier un bit en entrée peut modifier tous les bits de la sortie

2.6. Quelques grandeurs

- Avec un processeur 4GHz : 2^{60} cycles en environ 9 ans i.e. 2^{60} / (4x10^{9})
- Un super-calculateur de 2018 développe 122,3 PetaFLOPS. 2^{60} en 9,427 secondes
- Pour une attaque force brute sur une clé de 128 bits : il lui faut 2^{68} fois plus de temps i.e. 6400 fois l'âge estimé de l'univers

2.7. Chiffrement par flots

- s'inspire du masque jetable
- remplacer la clé aléatoire du masque par un générateur de nombres pseudo-aléatoires. La racine devient la clé

2.8. Chiffrement par flots formellement

2 algos déterministes

- **Init(s,IV)**, avec **s** la racine et **IV** un vecteur d'initialisation : calcule l'état initial $\mathbf{st_0}$
- **GetBits($\mathbf{st_i}$)** calcule un (ou plusieurs) bits y ainsi que l'état $\mathbf{st_{i+1}}$

2.9. Mise en oeuvre

Générer n bits (ou blocs de bits)

```
def generer_blocs(s,iv,n):
    st=Init(s,iv)
    blocs=[]
    for i in range(0,n):
        (y,st)=GetBits(st)
        blocs = blocs + [y]
    return blocs
```

Ce code n'est pas exécutable

2.10. Protocole

```
def send(s,m):
    iv = genIV()
    blocs = generer_blocs(s,iv,len(m))
    for i range(0,len(m)):
        c[i]=m[i]^blocs[i]
```

```
    return iv+c

def receive(s,iv.c):
    blocs = generer_blocs(s,iv,len(m))
    for i range(0,len(m)):
        m[i]=c[i]^blocs[i]
    return m
```

Ce code n'est pas exécutable

2.11. Qualité d'un chiffrement par flot

La sécurité sémantique est garantie si le générateur est pseudo-aléatoire… de qualité cryptographique

- les suites de buts engendrées passent tous les tests statistiques
- si un attaquant connait tout ou partie de la suite de bits générés par GetBits, il est difficile de retrouver la clé utilisée (ou la paire **(s,iv)**)

2.12. RC4

- Inventé en 1987 par Ron Rivest
- A résisté de nombreuses années, mais **ne doit plus être utilisé** aujourd'hui
- Lui préférer Salsa20 ou encore Chacha20

2.13. RC4 Init

```
def rc4_init(k):
    for i in range(0,256):
        S[i]=i
    n = len(k)
    j=0
    for i in range(0,256):
        j=(j+S[i] + k[i%n])%256
        swap(S,i,j)
    return (S,0,0)
```

2.14. RC4 GetBits

```
def rc4_getbits(st):
    (S,i,j)=st
    i = (i+1)%256
    j = (S[i]+j)%256
    swap(S,i,j)
```

```
    t= (S[i]+S[j])%256
    return ((S,i,j),S[t])
```

2.15. Version pédagogique — Mini RC4

- Version générant des mots de 3 bits i.e. entiers compris entre 0 et 7
- Même version que précédemment en remplaçant les 256 par 8
- Un message à chiffrer doit d'abord être converti en octal puis chaque entier sera *xoré* avec les mots générés par **rc4_getbits** à la demande
- La conversion octale est donnée ci-dessous (la lettre B est codée par **0,1**)

	0	1	2	3	4	5	6	7
0	A	B	C	D	E	F	G	H
1	I	J	K	L	M	N	O	P
2	Q	R	S	T	U	V	W	X
3	Y	Z	a	b	c	d	e	f
4	g	h	i	j	k	l	m	n
5	o	p	q	r	s	t	u	v
6	w	x	y	z	0	1	2	3
7	4	5	6	7	8	9	.	

2.16. Application

- Le message suivant a été chiffré avec la clé **K=[1,6,6,4]**
- Ne pas oublier que la méthode de chiffrement a été la suivante :
  - Codage en octal de M → M'
  - Générer des entiers de 3 bits à partir de **K** et |**M'**|
  - xorer chaque entier de M' avec chaque entier généré chronologiquement
- Déchiffrez le :

```
0, 7, 1, 7, 2, 2, 3, 5, 4, 7, 7, 4, 5, 4, 5, 1, 7, 2, 5,
6, 6, 3, 2, 5, 5, 7, 6, 1, 1, 2, 7, 1, 5, 5, 0, 4, 4, 7,
6, 2, 4, 2, 0, 3, 1, 3, 1, 0, 4, 3, 1, 7, 1, 0, 3, 4, 0,
5, 0, 4, 7, 7, 6, 6, 0, 2, 3, 5, 2, 1, 7, 0, 2, 1, 7, 4,
0, 1, 6, 1, 3, 6, 0, 4, 5, 2, 1, 2, 6, 3, 1, 4, 2, 2, 4,
3, 7, 7, 5, 2, 7, 1, 5, 4, 4, 7, 3, 5, 3, 2, 6, 6, 0, 7,
6, 2, 5, 6, 2, 5, 4, 0, 0, 6, 5, 2, 0, 3, 3, 6, 0, 1, 1,
5, 7, 7, 6, 7, 2, 2, 5, 4, 5, 1, 5, 3, 6, 0, 4, 4, 5, 7,
1, 4, 5, 3, 6, 1, 1, 4, 2, 0, 2, 1, 4, 7, 7, 0, 7, 6, 0,
7, 3, 3, 7, 7, 0, 3, 2, 1, 7, 7, 4, 0, 3, 6, 0, 1, 5, 0,
2, 2, 7, 4, 2, 4, 6, 6, 4, 3, 1, 5, 1, 7, 1, 5, 5, 5, 0,
2, 6, 5, 2, 2, 1, 3, 5, 2, 5, 6, 1, 5, 1, 2, 6, 2, 5, 2,
6, 0, 1, 7, 4, 1, 2, 7, 4, 2, 0, 0, 5, 0, 1, 7, 1, 2, 4,
6, 3, 3, 6, 5, 1, 7, 4, 5, 2, 3, 1, 5, 5, 6, 0, 4, 5, 3,
6, 4, 0, 4, 2, 2, 1, 4, 7, 4, 0, 7, 2, 3, 5, 6, 0, 1, 1,
```

```
4, 1, 1, 3, 2, 7, 3, 2, 2, 6, 6, 3, 3, 6, 2, 2, 4, 5, 5,
0, 1, 0, 2, 6, 5, 6, 4, 4, 7, 4, 3, 4, 2, 6, 2, 3, 3, 2,
5, 7, 5, 3, 2, 6, 1, 6, 6, 4, 2, 3, 5
```

2.17. Chiffrement par blocs

- Plutôt que de chiffrer des messages de tailles arbitraires, on se concentre sur des blocs de taille fixe
- Un message clair est alors scindé en blocs et chaque bloc subit des transformations
- Chaque clé possible doit générer une permutation proche de l'aléatoire

2.18. DES

- Développé par IBM
- Adopté en 1977 par le NBS
- DES n'est plus considéré comme sûr car clés trop petites (56 bits)

2.19. Chiffrement de Feistel

Idée

- Effectuer successivement plusieurs chiffrements simples
- La composition de chiffrements simples permet d'approcher une permutation quelconque des bits
- 1973 : Feistel propose une structure générique pour les chiffrements par blocs

2.20. Ronde de Feistel

```
def ronde_feistel(Mi,Ki,f= lambda x y :x^y ):
    Li.Ri=Mi
    assert len(Li)==len(Ri)=n
    return Ri.(Li^f(Ri,Ki))
```

- Plusieurs rondes à effectuer
- Autant de clés générées que de rondes à partir d'une clé originale
- En résumé :

  $L_{i+1}.R_{i+1}$=ronde_feistel($L_i.R_i,K_i$)

- f est une fonction qui effectue une opération à partir de deux blocs de tailles identiques pour fournir un bloc de même taille

2.21. Déchiffrement de Feistel ?

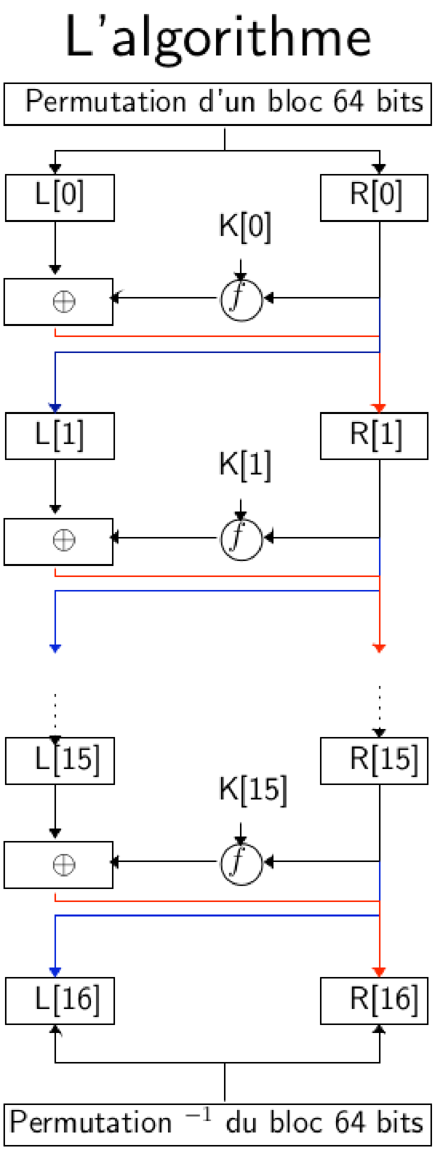
- Il faut le procédé inverse
- Application des clés dans le sens inverse i.e. Kp,…, K0
- On prend le même algorithme mais on inverse les blocs en entrée

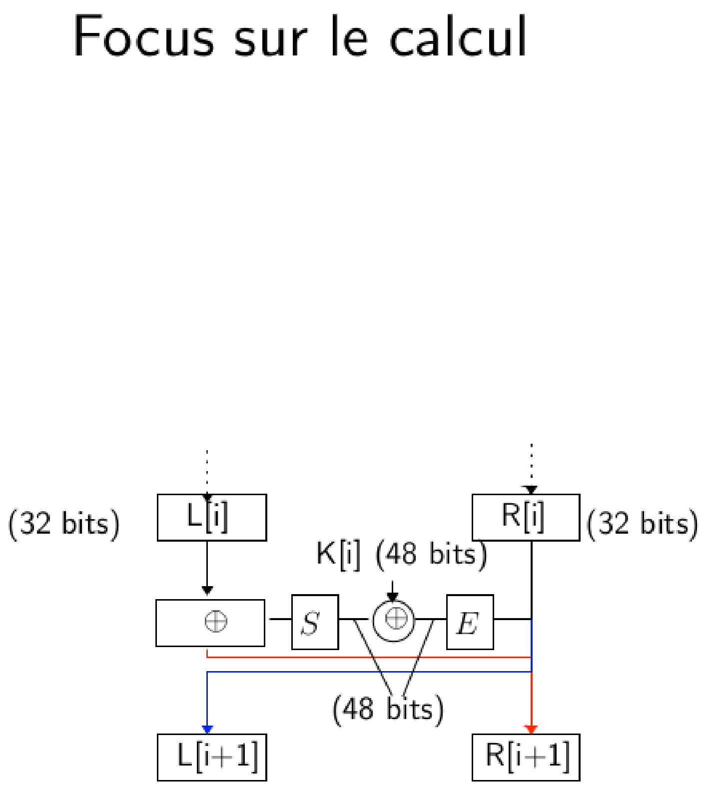
$L_i.R_i$=ronde_feistel($R_{i+1}.L_{i+1},K_{i+1}$)

2.22. DES - principe

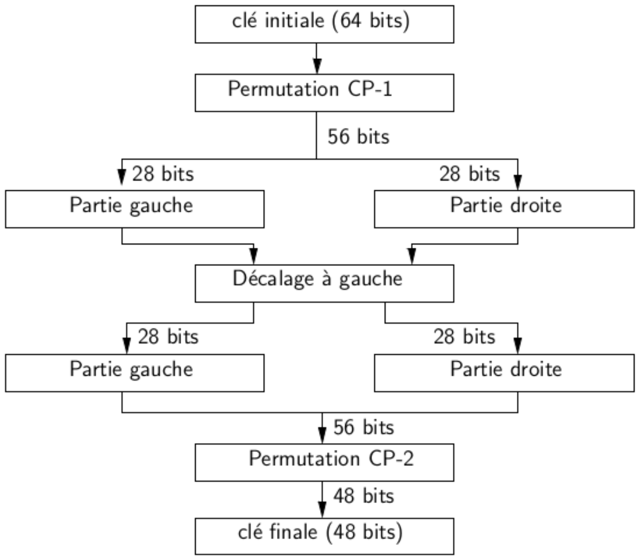
- Chiffrement par blocs de 64 bits
- Clé de 64 bits mais uniquement 56 bits sont utilisés

2.23. DES - Schéma





2.24. DES - Calcul des clés



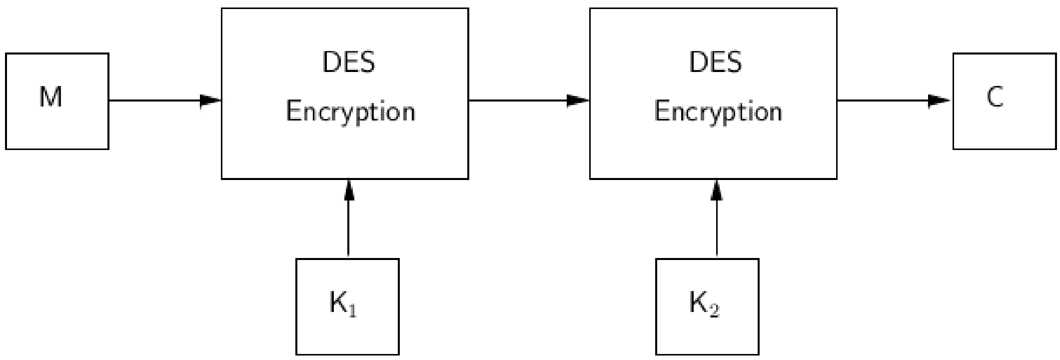
CP-1	57	49	41	33	25	17	9	1	58	50	42	34	26	18
	10	2	59	51	43	35	27	19	11	3	60	52	44	36
	63	55	47	39	31	23	15	7	62	54	46	38	30	22
	14	6	61	53	45	37	29	21	13	5	28	20	12	4

CP-2	14	17	11	24	1	5	3	28	15	6	21	10
	23	19	12	4	26	8	16	7	27	20	13	2
	41	52	31	37	47	55	30	40	51	45	33	48
	44	49	39	56	34	53	46	42	50	36	29	32

2.25. DES - déchiffrement

- DES est un chiffrement symétrique
- Comme pour Feistel, il faut inverser l'ordre des clés ainsi que les moitiés de message en entrée de ronde

2.26. DES double

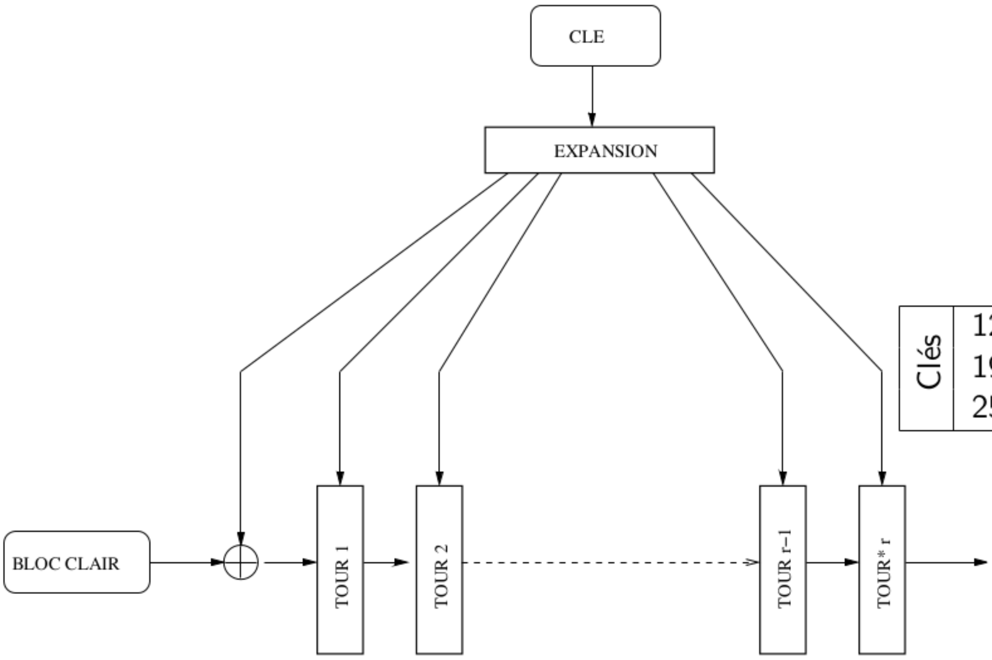


- Est ce équivalent à la sécurité d'une clé de taille de 112 bits ?
- La réponse est non i.e. attaque par RDV

2.27. Le standart actuel ? AES

- Système proposé par Joan Daemen et Vincent Rijmen
- Système de chiffrement symétrique
- A l'origine par blocs de 128, 192 ou 256 bits
- Clés de de 128, 192, 256 bits

2.28. AES - schéma principal



		Blocs		
		128	192	256
Clés	128	10	12	14
	192	12	12	14
	256	14	14	14

2.29. AES - Blocs ?

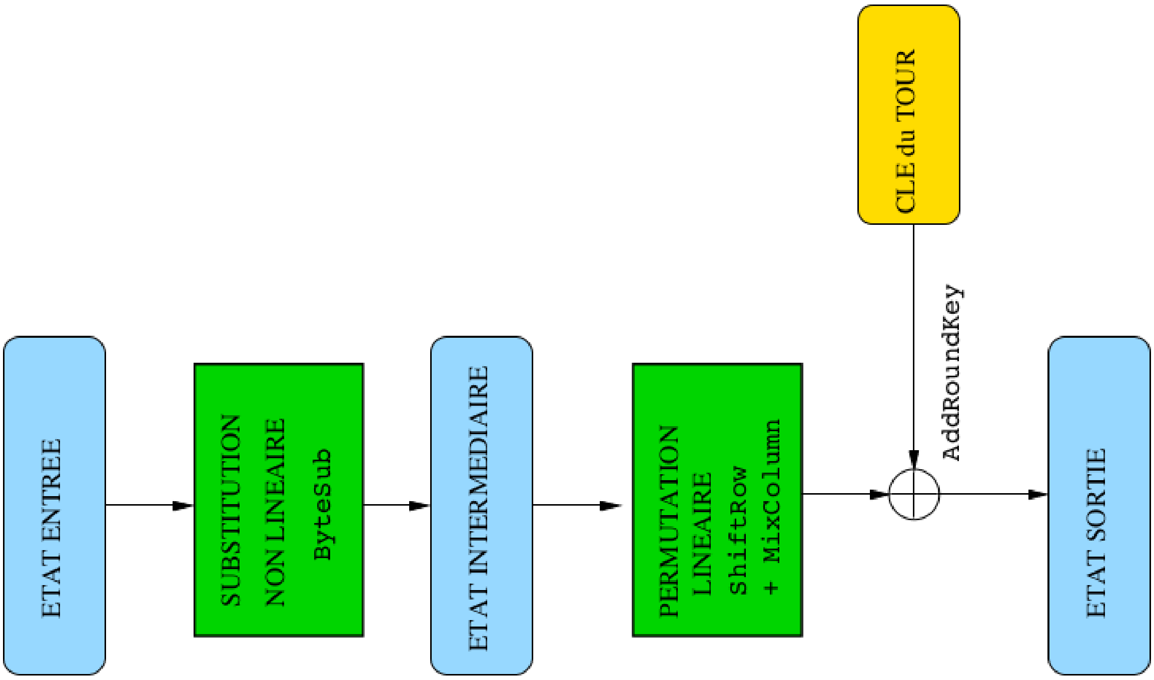
1 octet par case = un polynôme de degré 7

$$\begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix}$$

$X^6 + X^4 + X^2 + X^1 + 1 = 01010111 = 0x57$

2.30. AES - Transformations successives d’un état

- Pas de schéma de Feistel ici
- Schémas de substitutions et de transformations sur l’état entier



2.31. En résumé

- Il devrait y avoir des multiplications modulaires dans un corps de Galois
- ça se limitera finalement à des opérations sur des octets :
    - multiplications de nombres compris entre 0 et 255 qui donnent des nombres compris entre 0 et 255
    - additions de nombres compris entre 0 et 255 qui donnent des nombres compris entre 0 et 255
- En réalité, nous manipulerons une version pédagogique avec des blocs de 4 bits

2.32. Détails des opérations sur les états

- **ByteSub** : c'est une transformation qui est finalement une substitution d'octets par d'autres. Elle est appliquée sur chaque octet de la matrice
- **ShiftRow** : c'est un décallage circulaire vers la gauche appliqué sur la matrice selon la ligne. La première ligne n'est pas décalée. La seconde, toutes les valeurs sont décalées d'un cran, la troisième de 2, etc
- **MixColumn** : multiplication matricielle avec une matrice pré-définie.

$$\begin{pmatrix} b_{0,j} \\ b_{1,j} \\ b_{2,j} \\ b_{3,j} \end{pmatrix} = \begin{pmatrix} \alpha & 1+\alpha & 1 & 1 \\ 1 & \alpha & 1+\alpha & 1 \\ 1 & 1 & \alpha & 1+\alpha \\ 1+\alpha & 1 & 1 & \alpha \end{pmatrix} \begin{pmatrix} a_{0,j} \\ a_{1,j} \\ a_{2,j} \\ a_{3,j} \end{pmatrix}$$

2.33. Dernière étape

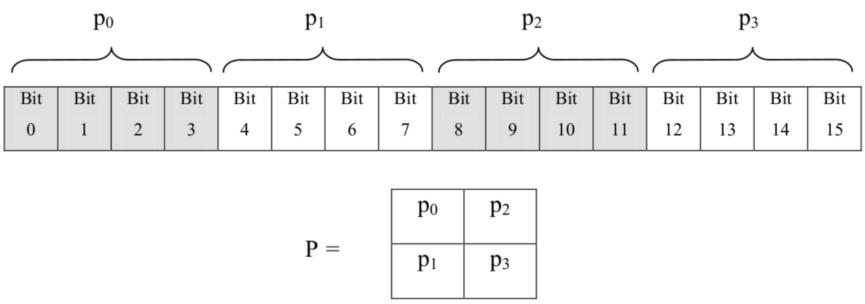
- XOR avec la clé de tour - 128 bits = 16*8
- Expansion de clés : 128 bits → 1408 bits (11 clé de 128 bits)
  - Phase un peu mystique avec une constante de tour pour chaque premier octet de nouvelle matrice
  - Plus de détails sur la version simplifiée

2.34. Déchiffrement AES

- Application de toutes les opérations inverses
- L’ordre des opérations est alors inversé
- L’application des clés est alors inversée

2.35. Mini AES

- Version pédagogique
- Mots et clés de 16 bits vu comme 4 nibbles

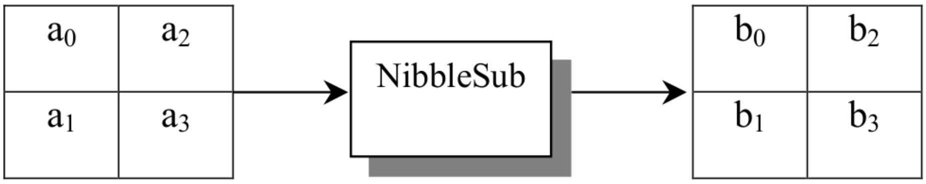


- 2 rondes

2.36. Corps de Galois

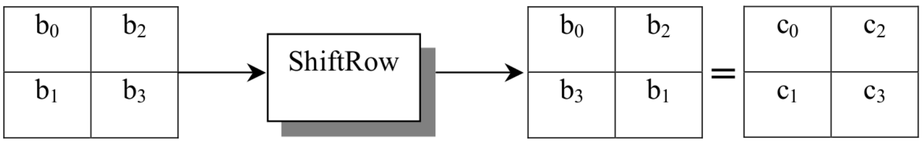
- Corps finis composés de tous les nibbles 0000, 0001, 0010,…, 1111 vus comme des polynômes à coefficients binaires 1101 = $x^3 + x^2 + 1$
- Addition : **xor**
- Multiplication : multiplication modulo x^4+x+1
- Exemple : $(x^3+ x + 1)*(x^2 + x + 1) = (x^5 + x^4 + 1) \bmod x^4+x+1 = x^2$

2.37. Mini AES — Nibble sub

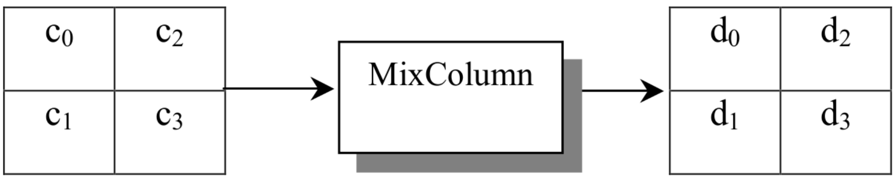


Input	Output	Input	Output
0000	1110	1000	0011
0001	0100	1001	1010
0010	1101	1010	0110
0011	0001	1011	1100
0100	0010	1100	0101
0101	1111	1101	1001
0110	1011	1110	0000
0111	1000	1111	0111

2.38. Mini AES — ShiftRow



2.39. Mini AES — MixColumn



$$\begin{bmatrix} d_0 \\ d_1 \end{bmatrix} = \begin{bmatrix} 3 & 2 \\ 2 & 3 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} 3 & 2 \\ 2 & 3 \end{bmatrix} \begin{bmatrix} c_2 \\ c_3 \end{bmatrix}$$

Calculs dans GF(2^4)

2.40. Mini AES — Addition de clé

$$\begin{array}{|c|c|} \hline d_0 & d_2 \\ \hline d_1 & d_3 \\ \hline \end{array} \oplus \begin{array}{|c|c|} \hline k_0 & k_2 \\ \hline k_1 & k_3 \\ \hline \end{array} = \begin{array}{|c|c|} \hline e_0 & e_2 \\ \hline e_1 & e_3 \\ \hline \end{array}$$

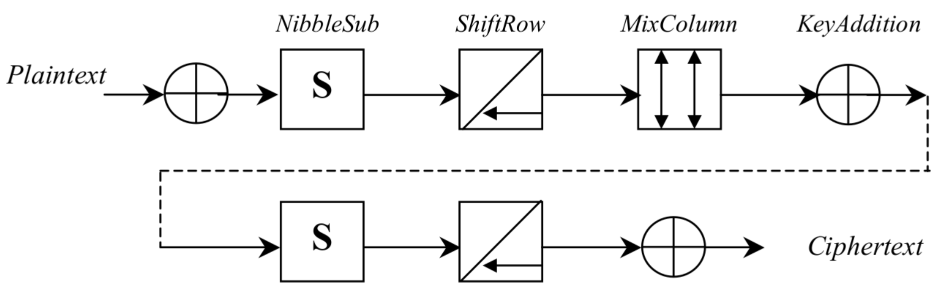
2.41. Mini-AES — les clés

- 2 rondes, 3 clés

Tour	Clé de tour
0	$w_0 = k_0,\ w_1 = k_1,\ w_2 = k_2,\ w_3 = k_3$
1	$w_4 = w_0 \oplus ByteSub(w_3) \oplus 0001,\ w_5 = w_1 \oplus w_4,$ $w_6 = w_2 \oplus w_5,\ w_7 = k_3 \oplus w_6$
2	$w_8 = w_4 \oplus ByteSub(w_7) \oplus 0010,\ w_9 = w_5 \oplus w_8,$ $w_{10} = w_6 \oplus w_9,\ w_{11} = w_7 \oplus w_{10}$

2.42. Mini AES — Rondes chiffrement

$$\text{Enc} = \left(\sigma_{K_2} \circ \pi \circ \gamma\right) \circ \left(\sigma_{K_1} \circ \theta \circ \pi \circ \gamma\right) \circ \sigma_{K_0}$$



2.43. Mini AES — Rondes déchiffrement

$$\begin{aligned}
\text{Dec} &= \left(\left(\sigma_{K_2} \circ \pi \circ \gamma\right) \circ \left(\sigma_{K_1} \circ \theta \circ \pi \circ \gamma\right) \circ \sigma_{K_0}\right)^{-1} \\
&= \sigma_{K_0}^{-1} \circ \left(\sigma_{K_1} \circ \theta \circ \pi \circ \gamma\right)^{-1} \circ \left(\sigma_{K_2} \circ \pi \circ \gamma\right)^{-1} \\
&= \sigma_{K_0}^{-1} \circ \left(\gamma^{-1} \circ \pi^{-1} \circ \theta^{-1} \circ \sigma_{K_1}^{-1}\right) \circ \left(\gamma^{-1} \circ \pi^{-1} \circ \sigma_{K_2}^{-1}\right) \\
&= \sigma_{K_0} \circ \left(\gamma^{-1} \circ \pi \circ \theta \circ \sigma_{K_1}\right) \circ \left(\gamma^{-1} \circ \pi \circ \sigma_{K_2}\right)
\end{aligned}$$

Input	Output	Input	Output
0000	1110	1000	0111
0001	0011	1001	1101
0010	0100	1010	1001
0011	1000	1011	0110
0100	0001	1100	1011
0101	1100	1101	0010
0110	1010	1110	0000
0111	1111	1111	0101

2.44. Autres

⊕	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
1	1	0	3	2	5	4	7	6	9	8	b	a	d	c	f	e
2	2	3	0	1	6	7	4	5	a	b	8	9	e	f	c	d
3	3	2	1	0	7	6	5	4	b	a	9	8	f	e	d	c
4	4	5	6	7	0	1	2	3	c	d	e	f	8	9	a	b
5	5	4	7	6	1	0	3	2	d	c	f	e	9	8	b	a
6	6	7	4	5	2	3	0	1	e	f	c	d	a	b	8	9
7	7	6	5	4	3	2	1	0	f	e	d	c	b	a	9	8
8	8	9	a	b	c	d	e	f	0	1	2	3	4	5	6	7
9	9	8	b	a	d	c	f	e	1	0	3	2	5	4	7	6
a	a	b	8	9	e	f	c	d	2	3	0	1	6	7	4	5
b	b	a	9	8	f	e	d	c	3	2	1	0	7	6	5	4
c	c	d	e	f	8	9	a	b	4	5	6	7	0	1	2	3
d	d	c	f	e	9	8	b	a	5	4	7	6	1	0	3	2
e	e	f	c	d	a	b	8	9	6	7	4	5	2	3	0	1
f	f	e	d	c	b	a	9	8	7	6	5	4	3	2	1	0

⊗	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
2	0	2	4	6	8	a	c	e	3	1	7	5	b	9	f	d
3	0	3	6	5	c	f	a	9	b	8	d	e	7	4	1	2
4	0	4	8	c	3	7	b	f	6	2	e	a	5	1	d	9
5	0	5	a	f	7	2	d	8	e	b	4	1	9	c	3	6
6	0	6	c	a	b	d	7	1	5	3	9	f	e	8	2	4
7	0	7	e	9	f	8	1	6	d	a	3	4	2	5	c	b
8	0	8	3	b	6	e	5	d	c	4	f	7	a	2	9	1
9	0	9	1	8	2	b	3	a	4	d	5	c	6	f	7	e
a	0	a	7	d	e	4	9	3	f	5	8	2	1	b	6	c
b	0	b	5	e	a	1	f	4	7	c	2	9	d	6	8	3
c	0	c	b	7	5	9	e	2	a	6	1	d	f	3	4	8
d	0	d	9	4	1	c	8	5	2	f	b	6	3	e	a	7
e	0	e	f	1	d	3	2	c	9	7	6	8	4	a	b	5
f	0	f	d	2	9	6	4	b	1	e	c	3	8	7	5	a

	γ	γ^{-1}
0	e	e
1	4	3
2	d	4
3	1	8
4	2	1
5	f	c
6	b	a
7	8	f
8	3	7
9	a	d
a	6	9
b	c	6
c	5	b
d	9	2
e	0	0
f	7	5

2.45. Comment chiffrer un message dont la taille supérieure à 16 bits ?

- 16 bits parce que mini-AES
- Mais la question est la même pour des blocs de 128 bits
- Pourquoi pas casser le message de départ en blocs de 16 bits et éventuellement compléter (padding) pour le dernier bloc
- Ensuite on chiffre **pour le moment** chaque bloc avec AES et la clé K
- Pourquoi c'est nul de faire comme cela ?
- Deux blocs identiques seront chiffrés de la même façon → ça sent les statistiques → ça sent un chiffrement pas parfait du tout

2.46. Pour ceux qui ont envie de s'amuser

- Le message ci-dessous a été chiffré avec le mini-AES bloc par bloc

```
3987 50f9 b6d6 4c06 9f02 29df 7c05 ec10 d983 9322 fb43 389d 6f19 368d d983
b0f0 c878 2c1e 9ea2 bf16 60f8 5422 0d6a ed3c 6328 368d 6d69 c0f7 e9d0 9f39
a0ca 5d02 66d9 c0f7 5329 bb41 b511 ed0c d983 eaf0 356d 098b d0fe c878 ff33
3e5d ac0a c327 66d9 5329 ed30 7c05 9322 bf16 6519 d0c3 c0f7 5329 def3 ec10
d193 ff34 1d0e b976 bc06 9f02 df33 098b d0fe bc06 a1da 3d3d 26de 3d3d 6b49
9f39 374d 56d2 6f39 9e52 3987 60f8 c258 151e 6b49 80f6 3e5d f0f3 1f3f 119f
5422 60f8 5d02 e560 7c05 ec10 d983 6328 f1d4 80f6 ec10 d983 60f8 c258 151e
6b49 80f6 3e5d f0f3 1f3f 119f 5422 6328 f1d4 80f6 4886 f0ff 3d3d 0d6a c428
29de
```

- On sait que la clé utilisée est : fee7
- On a utilisé la table ci-dessous pour encoder les caractères dans une premier temps sur 8 bits

20	21	2C	2E	30	31	32	33	34	35	36	37	38	39	41	42	43	44	45	46	47	48	49	4A	4B	4C
SP	!	,	.	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	G	H	I	J	K	L
4D	4E	4F	50	51	52	53	54	55	56	57	58	59	5A	61	62	63	64	65	66	67	68	69	6A	6B	6C
M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	a	b	c	d	e	f	g	h	i	j	k	l
6D	6E	6F	70	71	72	73	74	75	76	77	78	79	7A												
m	n	o	p	q	r	s	t	u	v	w	x	y	z												

- Indice sur le procéssus de chiffrement : Yeah → 59 65 61 68 → 5965 6168 (2 blocs de 16 bits)

2.47. Bilan sur l'utilisation d'AES

- Il semble évident qu'AES brouille bien les pistes d'un point de vue bloc
- Par contre, il faut revoir l'utilisation qu'on en fait sur des messages plus grands que la taille prévue
- Besoin de brouiller dans une plus grande généralité les blocs chiffrés produits

2.48. Mise en application d'AES

- Le chapitre suivant parle de modes de chiffrement
- On part du principe ici qu'un message est cassé en blocs d'une taille donnée i.e. 128 bits
- Chaque bloc est chiffré avec la même clé
- On appelle ce mode ECB (Electronic Codebook Block)
- Le contenu du fichier à l'URL ci-dessous est la représentation hexadécimale du binaire chiffré
  - https://celene.univ-orleans.fr/pluginfile.php/1584046/mod_resource/content/4/defi.txt
- La clé utilisée est : ec51b7aa64b2b885e05af105ba929d69bd20a52b103b26bccd7f633433cc1c04

Chapter 3. Modes opératoires

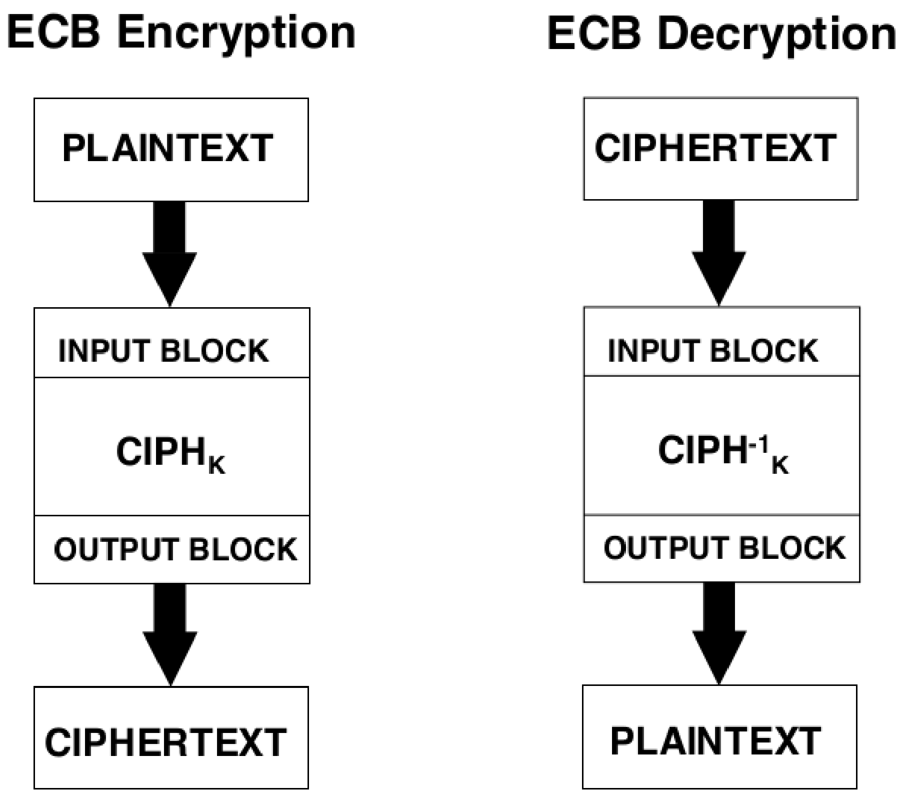
3.1. Previously in Desperate Student’s Life

- Algo standart symétrique permettant de chiffrer des blocs de bits
- Faiblesse car approche naïve donne pour deux blocs clairs identiques, deux blocs chiffrés également identiques
- Comment s’en sortir ?

3.2. Modes opératoires

- Pour le chiffrement par flot, on génére autant de bits que nécessaire pour la clé
- Pour le chiffrement par blocs, même idée à un détail près… la clé est de taille fixe

3.3. ECB (electronic codebook)



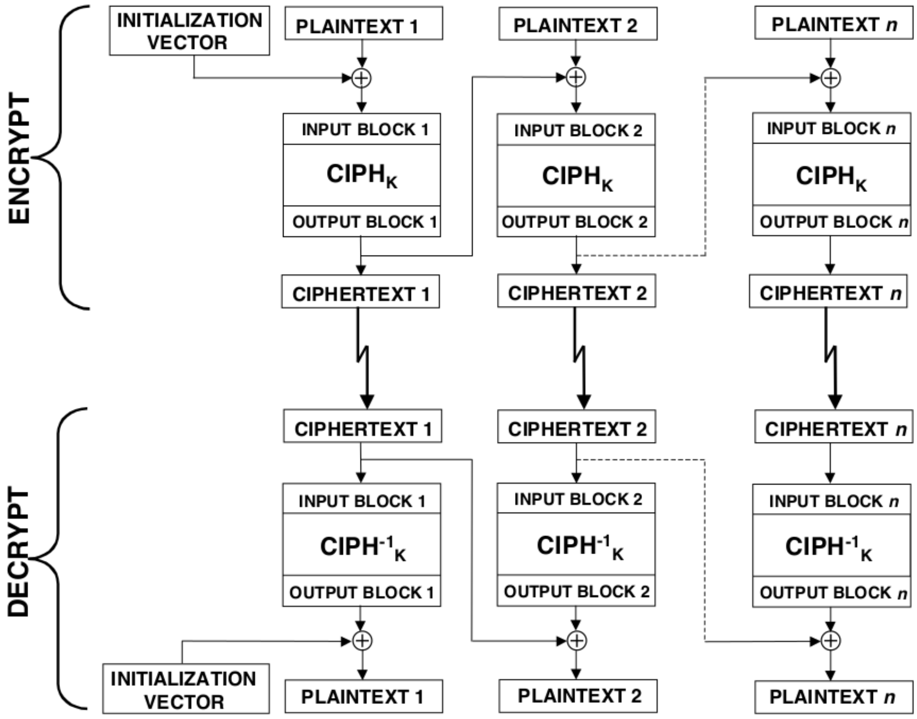
Deux blocs identiques sont chiffrés de la même façon

3.4. Padding

- Pour les modes ECB, CBC, CFB il est nécessaire de compléter le message clair
- Le padding doit être réversible
  - train de bits 100000…000

◦ séquence de k octets de valeur k (PKCS)

3.5. CBC (Cypher Block Chaining)



3.6. Application

- Le message suivant a été chiffré avec :
  ◦ AES-CBC
  ◦ la clé b33f

```
5eed 0e89 dde4 b1b9 aafa c880 8de5 a72d ce4e cb6c
bf7d ef6b 8b36 1d23 969e 1427 20d3 2794 bcc1 40cb
da8b 1a7b 6aa4 9fe4
```

3.7. Indices

Voici les tables AES précalculées pour la clé **b33f**

m	0385	1a88	25b9	45f3	4aed	6de1	7255	7318
$F_{b33f}(m)$	9fe4	0e89	da8b	2794	bcc1	dde4	20d3	6aa4

m	7e4b	8d60	adee	b482	bfab	cf0f	cf97	cfb5
$F_{b33f}(m)$	969e	cb6c	8de5	b1b9	1a7b	8b36	c880	40cb

m	d7cb	de0e	e46f	eb3c	ee16	f9c5	fff8
$F_{b33f}(m)$	aafa	ef6b	ce4e	bf7d	1d23	a72d	1427

3.8. Indices

20	21	2C	2E	30	31	32	33	34	35	36	37	38	39	41	42	43	44	45	46	47	48	49	4A	4B	4C
SP	!	,	.	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	G	H	I	J	K	L
4D	4E	4F	50	51	52	53	54	55	56	57	58	59	5A	61	62	63	64	65	66	67	68	69	6A	6B	6C
M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	a	b	c	d	e	f	g	h	i	j	k	l
6D	6E	6F	70	71	72	73	74	75	76	77	78	79	7A												
m	n	o	p	q	r	s	t	u	v	w	x	y	z												

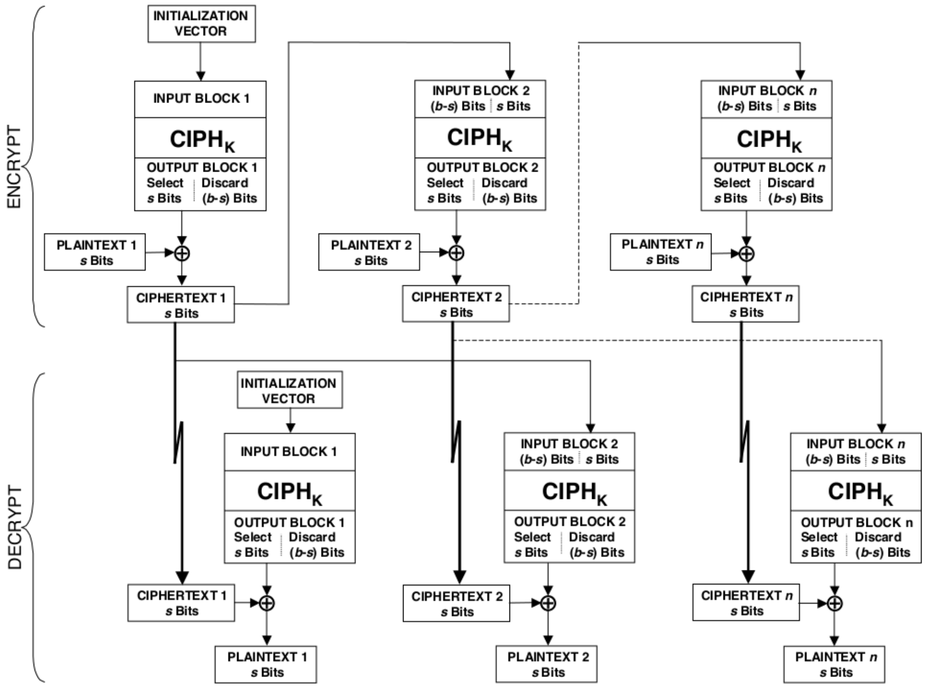
3.9. Indices

x^y	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
1	1	0	3	2	5	4	7	6	9	8	b	a	d	c	f	e
2	2	3	0	1	6	7	4	5	a	b	8	9	e	f	c	d
3	3	2	1	0	7	6	5	4	b	a	9	8	f	e	d	c
4	4	5	6	7	0	1	2	3	c	d	e	f	8	9	a	b
5	5	4	7	6	1	0	3	2	d	c	f	e	9	8	b	a
6	6	7	4	5	2	3	0	1	e	f	c	d	a	b	8	9
7	7	6	5	4	3	2	1	0	f	e	d	c	b	a	9	8
8	8	9	a	b	c	d	e	f	0	1	2	3	4	5	6	7
9	9	8	b	a	d	c	f	e	1	0	3	2	5	4	7	6
a	a	b	8	9	e	f	c	d	2	3	0	1	6	7	4	5
b	b	a	9	8	f	e	d	c	3	2	1	0	7	6	5	4
c	c	d	e	f	8	9	a	b	4	5	6	7	0	1	2	3
d	d	c	f	e	9	8	b	a	5	4	7	6	1	0	3	2
e	e	f	c	d	a	b	8	9	6	7	4	5	2	3	0	1
f	f	e	d	c	b	a	9	8	7	6	5	4	3	2	1	0

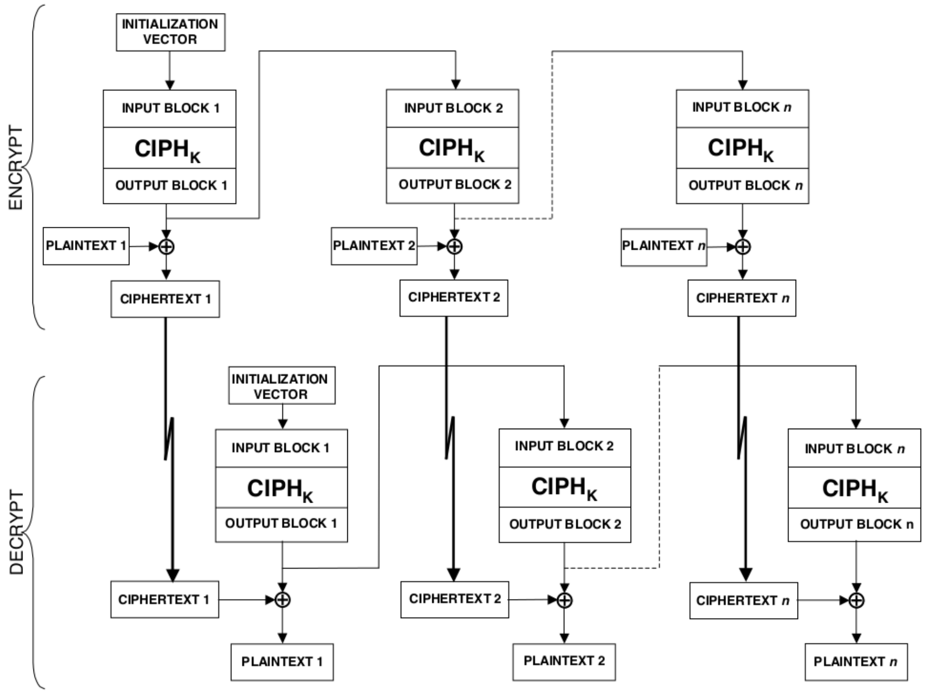
3.10. Vecteur d’initialisation

- Pour éviter deux premiers blocs identiques soient chiffrés exactement de la même façon, on ajoute un vecteur d’initialisation
- L’IV n’a pas besoin d’être secret
- L’IV ne doit pas être prédictible
    - Générateur pseudo-aléatoire de qualité cryptographique
    - Chiffrer une suite prédictible

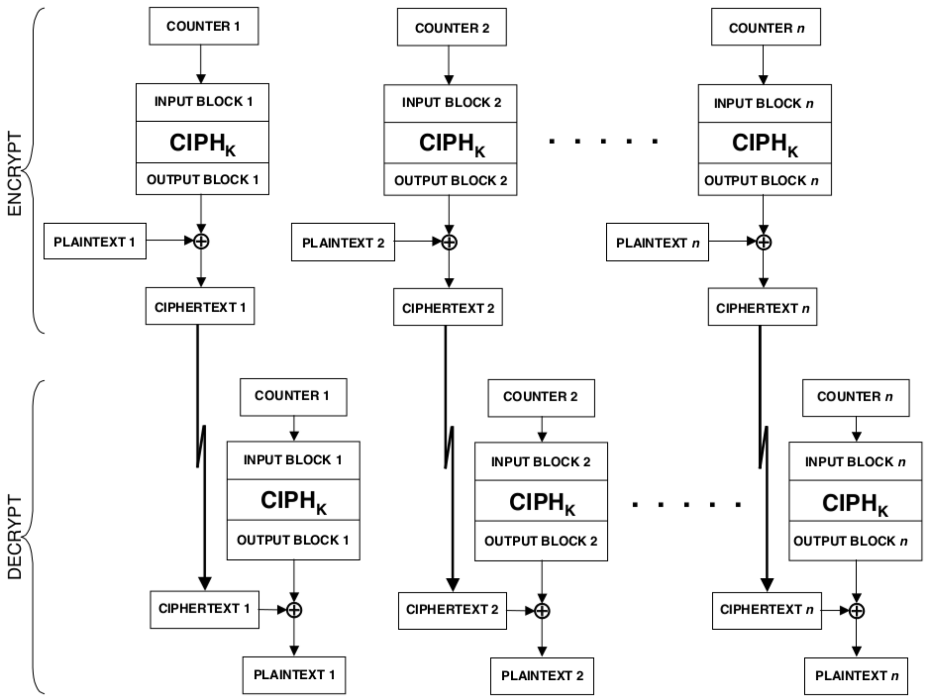
3.11. CFB (Cypher Feed Back)



3.12. OFB (Output Feed Back)



3.13. CTR (Counter)



3.14. Génération de compteurs

- CTR nécéssite une suite de blocs compteurs à usage unique
- Incrémenter le compteur pour obtenir une suite de blocs distincts

3.15. Application

- On sait que le message ci-dessous a été chiffré en CTR avec la clé b33f
- Toutes les données liées à AES sont données dans le slide suivant
- Voici le message à déchiffrer

```
5eed 4450 b356 9952 5479 d76e 1766 d627 d15d 302f 6252 837c
0268 772d a16e eb66 3478 6725 bf77 8170 5e49 de13 125b c217
f556 0511 3b5e 9e4c 0b1d
```

3.16. Tables de calculs liées à AES dans CTR

Voici les tables AES précalculées pour la clé **b33f**

m	5eed	5eee	5eef	5ef0	5ef1	5ef2	5ef3	5ef4
$F_{b33f}(m)$	0035	d03e	f034	320b	b203	7208	a207	9209

m	5ef5	5ef6	5ef7	5ef8	5ef9	5efa	5efb	5efc
$F_{b33f}(m)$	6201	4202	e20f	220c	120d	c206	8200	520a

m	5efd	5efe	5eff	5f00	5f01	5f02	5f03	5f04
$F_{b33f}(m)$	0205	d20e	f204	3b3b	bb33	7b38	ab37	9b39

m	5f05	5f06	5f07	5f08
$F_{b33f}(m)$	6b31	4b32	eb3f	2b3c

3.17. CTR est fragile

- Voici une discussion interceptée en AES-CTR avec la même clé de chiffrement utilisée à chaque fois.
- Indice : d'après une source sûre, le premier message commence par : "Bonjour Nicolas !"
- Essayez de percer le mystère !!

```
Yoh:f01e57228b262787dad69692eb9034910bde29d1cc8218911bd8d980e890978c5a9b7689c8983f3
2c073af3bec76363008717f76ff37
Nicolas:f01e462c89393cd2f199b09ae6d178bf0d9724d1f19b1892098dccd5bc81978c5e9f6add849
96a7cc57bb869ef3f3769096ae263ff37
Yoh:f01e566a803f3cd24f97f6dbc49a789e17936a83fdd7558d1b8c5087add39d8a5cda7b92858c256
14336be7eaa277a7312776c77e872
Nicolas:f01e56228a2066d2e980b995fcdf3b950cde6d89fd855b9d0b9d94d5a19fd880089f76dd89d
c297dc774b37ee43f
Yoh:f01e566a803f3cd2c4d1b98de9912cd01c9b7a9ff1924ad488d8cc87a7868e9c5ad438a99ddc3c7
7df6efa77ef3f397815777978e837
Nicolas:f01e5a26c96c24978895b092ee992a95589b66d1e8984b9d1c91d79be8cbd88a4d8879dd8c9
324718a7abf3bbf3f
Yoh:f01e5a06c96ca8d2cd83a0dbec9a789a178b6d83b8d6
Yoh:f01e5c21c53e2d81dc93f81ba88b2a9f0d886d83b89b5dd40b90d193ae819dd94d94388d878f236
6c379b43bbd31
```

3.18. Comparaison des modes opératoires

Les critères de comparaison

- Auto-synchronisation
- Accès randomisé
- Chiffrement ou déchiffrement parallélisable
- Capacité à supporter des erreurs

3.19. Quel algorithme choisir ?

- Pour garantir la confidentialité
  - AES-256-CBC avec IV aléatoires
  - AES-256-CTR avec nonce distincts
- Attention la confidentialité ne suffit pas toujours !!!

3.20. Quelques exemples en pratique : Python

Utilisation de la bibliothèque pycryptodome

```
key = get_random_bytes(32)
iv = get_random_bytes(16)

# Message à chiffrer
message = b"Hello, World!"

# Création d'un objet AES en mode CBC avec la clé et le vecteur d'initialisation
cipher = AES.new(key, AES.MODE_CBC, iv)
# Chiffrement du message avec padding
ciphertext = cipher.encrypt(pad(message, AES.block_size))

cipher = AES.new(key, AES.MODE_CBC, iv)
decrypted_message = unpad(cipher.decrypt(ciphertext), AES.block_size)
```

3.21. Quelques exemples en pratique : Java

Utilisation de la bibliothèque Bouncy Castle

```
public class ProgrammeExtrait {

    private static final int keyLength = 32;
    private static final SecureRandom random = new SecureRandom();

    public static void main(String [] args) throws Exception {
        Security.addProvider(new BouncyCastleProvider());

        String plaintext = "hello world";
        String ivStr = "0123456789abcdef";
        SecretKey secretKey = generateKey();
        byte [] ciphertext = encrypt(secretKey,ivStr,plaintext);
        String recoveredPlaintext = decrypt(secretKey,ivStr,ciphertext);

        System.out.println(recoveredPlaintext);
    }


    private static byte [] encrypt(SecretKey key, String ivStr,String plaintext)
throws Exception {
        Cipher cipher = Cipher.getInstance("AES/CBC/PKCS7Padding");
        byte[] iv = ivStr.getBytes("US-ASCII");
        cipher.init(Cipher.ENCRYPT_MODE, key, new IvParameterSpec(iv));
        return cipher.doFinal(plaintext.getBytes());
    }

    private static String decrypt(SecretKey key, String ivStr,byte [] ciphertext)
throws Exception {
        Cipher cipher = Cipher.getInstance("AES/CBC/PKCS7Padding");
        byte[]iv = ivStr.getBytes("US-ASCII");
```

```java
        cipher.init(Cipher.DECRYPT_MODE, key, new IvParameterSpec(iv));
        return new String(cipher.doFinal(ciphertext));
    }

    private static SecretKey generateKey() throws Exception {
        byte[] keyBytes = new byte[keyLength];
        random.nextBytes(keyBytes);
        SecretKeySpec keySpec = new SecretKeySpec(keyBytes, "AES");
        return keySpec;
    }
}
```

3.22. Le plus compliqué dans l’histoire

- Ce n’est pas forcément d’utiliser les algos
- C’est plutôt de manipuler les bases azotées de l’informatique : octets
- D’ailleurs, sauriez vous déchiffrer les octets ci-dessous ? (indice : le résultat sera un fichier et on fonctionne en CBC)

```
Clé utilisée : 692cec2b449bf78f1fb520f8fa93a703404aad93cf2c0f5e983a9dee4930d6d5
Vecteur d'initialisation (IV) utilisé : a153a44bf7826505ef8c654677bfc26f

5f280a81621cff1401dabd8c7faa0d33555c5e0c3838ef605ab6c81e5abb146f3e06c267058a51f0831
3b1add495d62c
fc26c3f76e699b5be755093ee4efbf3d381d9f0737c55f535e8aef16f484ece2adba3d2684816c9d63c
28039f3ff95e3
b6bcbfc00bfe8161770846f2ea745915613d40ba66f2f3c848750849484931700d9a04861650e2f8893
d67908c63740d
12e993e8a65152e3afdefdde67d7550a8d5d8aaa091efd2237f3f3e2e9af89cc93569f5a14307b76b94
7722ebbf91c32
4860d55e271ae6a06e0e90f4f3f617de4fa3fe953eb01d59f28cacdacdd4871970746187b9837dd0b06
2fb3a7ac730dc
62a866df586306ec979680edbbc413caab6e195ec2e951e174447998e01a7c070d3701b5d93b1486bdb
ba1ed76a053e2
e73e531e3121c09ec511e62c64ac6a5fdc1f9a831901cd02e72812065c84d72ddbb59a2d73d7b934a63
3322bed997a69
110282572756b6d53556512154cd11ed537d74185bd13e364f30d425bbaa89ca
```

3.23. Et si les algos étaient parfaits ?

- ça ne suffit pas pour autant
- Protocoles cryptographiques sont fondés à partir des primitives cryptographiques
- Les propriétés attendues sont en général :
  - Confidentialité
  - Intégrité

    - Authenticité
    - Non répudiation

3.24. Bilan sur les primitives cryptographiques jusqu’à présent

- **Cryptographie symétrique (à clé secrète)**
- Cryptographie asymétrique (à clé publique)
- Fonctions de hachage de qualité cryptographique
- Codes d’authentification de messages (MAC)
- Algorithmes de signature numérique

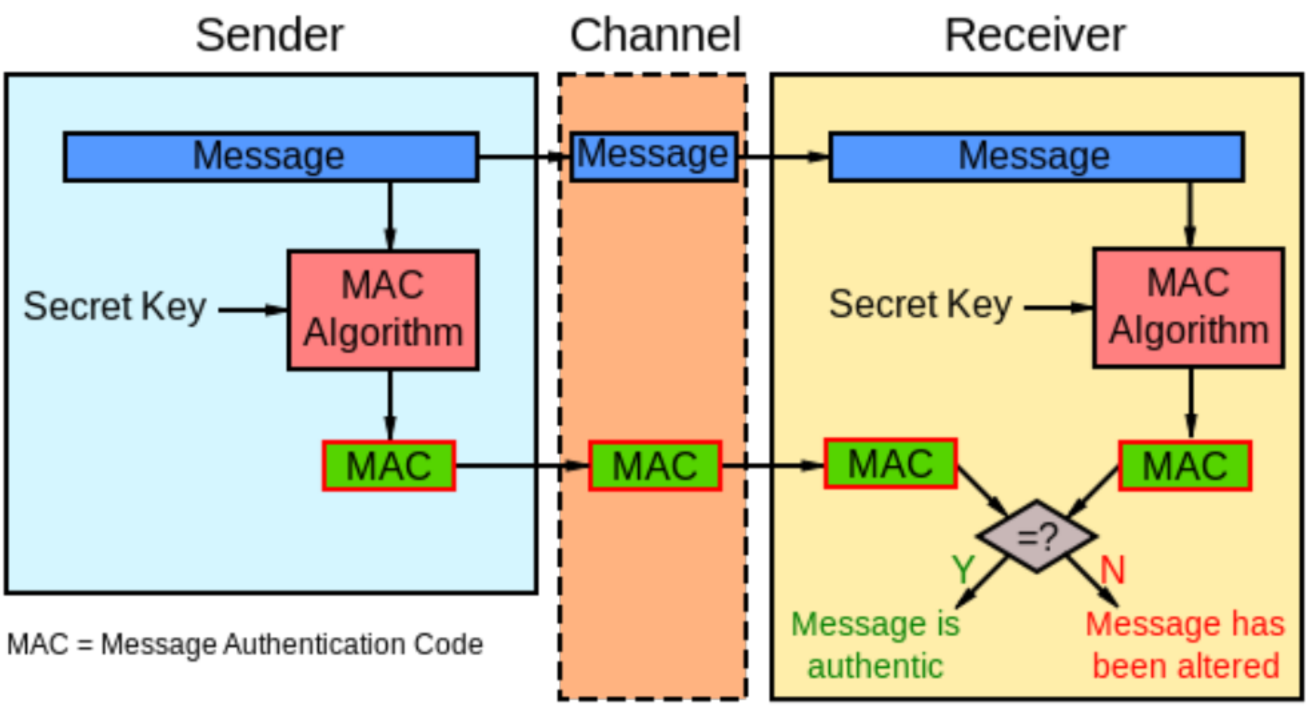
Chapter 4. Intégrité des messages

4.1. Previously in Desperate Student’s Life

- Algo standart symétrique permettant de chiffrer des blocs de bits
- Des modes de chiffrement pour casser les analyses statistiques
- Finalement, comment peut on être sûr que :
  - Le message provient d’une personne supposée
  - Si tel est le cas, comment être sûr que le message n’a pas été modifié entre temps
- Après tout, un message est une suite de bits

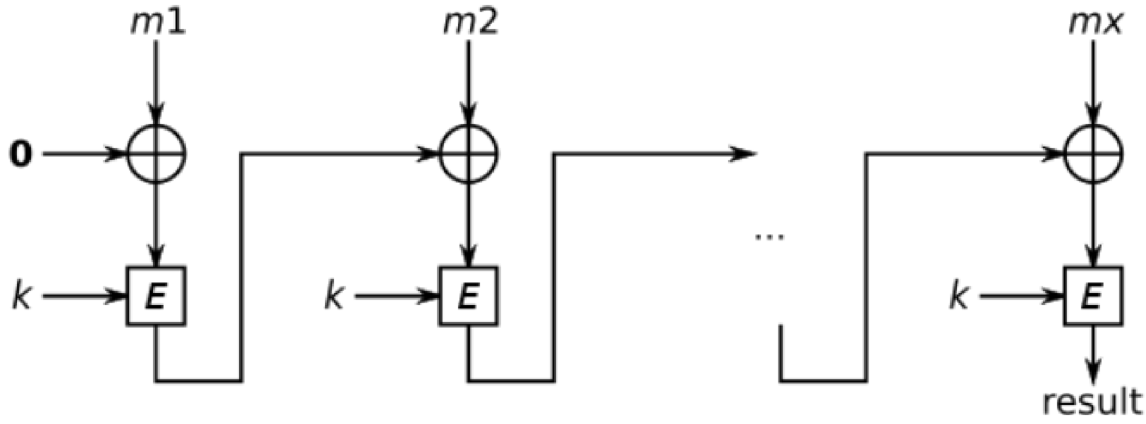
4.2. MAC

Message Authentication Code



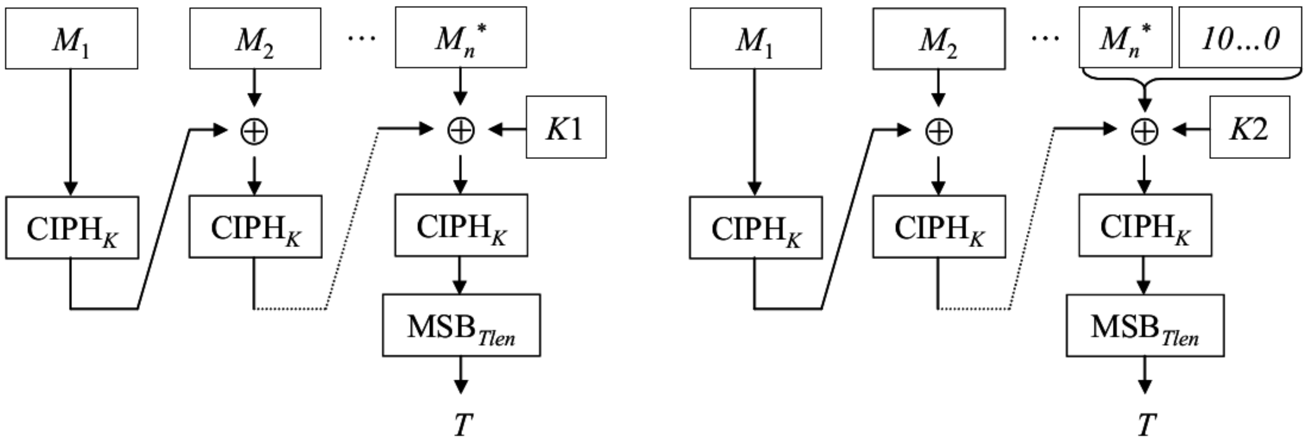
4.3. CBC-MAC

- Utiliser le dernier bloc d’un chiffrement comme MAC (avec une autre clé)



- Possibilité de forger des messages valides pour des messages de taille variable → CBC-MAC non sûr pour les messages de taille variable

4.4. Amélioration CMAC - NIST800-38B



4.5. Pause exercice

- Alice a la brillante idée de combiner AES-256- CBC avec AES-CBC-MAC.
- Munie d’une clé secrète K de 256 bits, elle chiffre un message M comme suit :

$$E_K(M) \parallel H_K(M)$$

FBI ou non ?

4.6. Hachage

- H: $\{0,1\}^* \rightarrow \{0,1\}^t$
- Fonction qui associe une empreinte de taille fixe à une entrée de taille arbitraire
- Tables de hashage
- Propriété attendue : les valeurs prises par H doivent être uniformément réparties ; faible

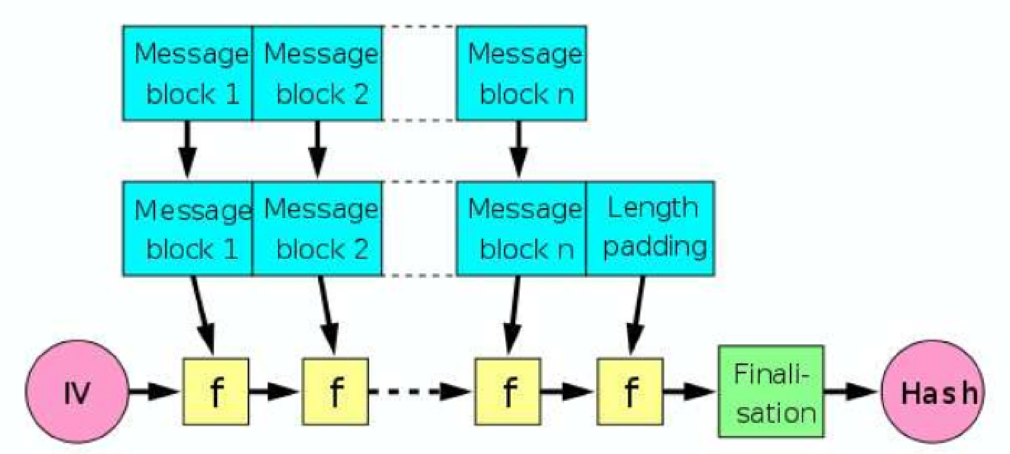
probabilité de collision H(m)=H(m')

4.7. Hachage cryptographique — propriétés attendues pour H

- one-way : difficile de retrouver x à partir de H(x)
- résistance aux collisions : difficile de trouver x et y tels que H(x)=H(y)
- Indiscernable d'une fonction aléatoire

4.8. Construction Merkle-Damgard

$$
\begin{aligned}
m &= m_1 \| m_2 \| \cdots \| m_k \\
H_0 &= \mathrm{IV} \\
H_i &= F(H_{i-1}, m_i) \\
H(m) &= G(H_k)
\end{aligned}
\qquad
\begin{aligned}
F &: \{0,1\}^s \times \{0,1\}^b \to \{0,1\}^s \\
G &: \{0,1\}^s \to \{0,1\}^t
\end{aligned}
$$



4.9. Pause exercice

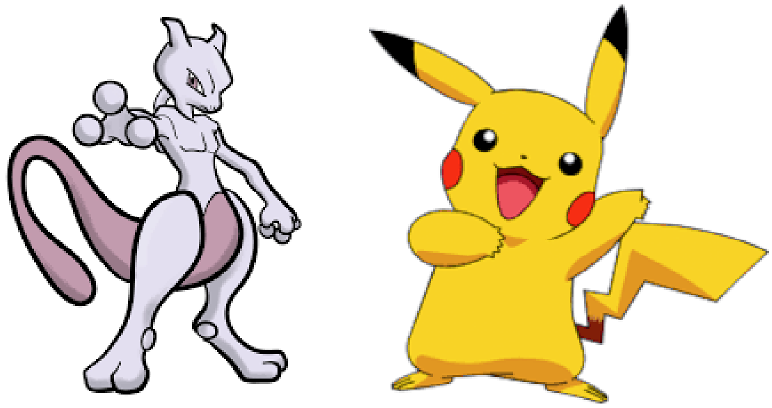
- Charli trouve que la construction MD ressemble beaucoup au chaînage CBC de la semaine dernière.
- Il propose de prendre IV=0 et comme fonction de compression le chaînage d'AES-256 avec la clé 0.
- Montrer que cette fonction n'est pas du tout résistante aux collisions !

4.10. MD5 (RFC 1321)

- Inventé par Rivest en 1992
- Très populaire malgré une première faille en 1995
- Empreinte de 128 bits (blocs de 512 bits)

- Conçu pour être rapide sur architecture 32 bits
- Padding : on ajoute un bit à 1 puis des 0 pour obtenir une taille congrue à 448 mod 512 et enfin on ajoute la longueur initiale du message codée sur 64 bits.

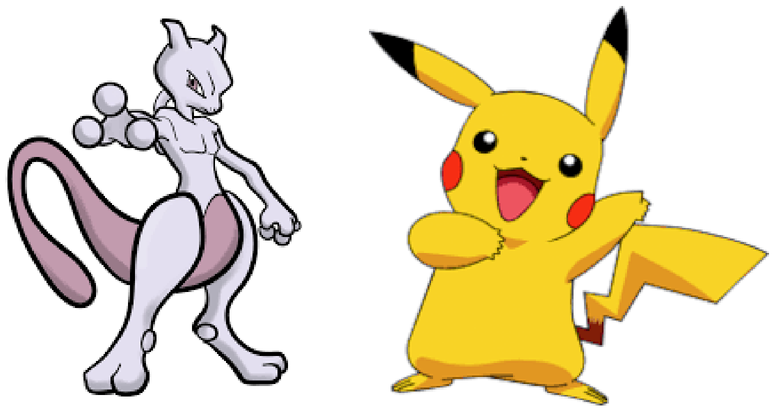
4.11. Application MD5 sur images



```
md5sum *.png
c23bdb95d34d1d56c8a5e6845182b8b1  pika.png
d7df07f1874f3631632feeedd3cb869a  pokemonmew2.png
```

4.12. Mais...

Modifions quelque peu les images avec https://github.com/corkami/collisions/blob/master/scripts/png.py



```
md5sum collision*.png
4905e947e3b9542011ab2c1e8721a78f  collision1.png
4905e947e3b9542011ab2c1e8721a78f  collision2.png
```

4.13. MD5 suite des ennuis

- On peut faire la même chose sur des programmes python, des PDF, etc
- Toutes les attaques sont basées sur : MD5(xyz) = MD5(xy'z)

- Le **z** dans le cas des PNG : concaténation des deux images mais l'interprétation globale donnera soit l'une, soit l'autre
- Pour les curieux
  - https://www.youtube.com/watch?v=BcwrMnGVyBI
  - https://github.com/corkami/

4.14. MD5 — the end

4.15. SHA-1 et SHA-2

- Développé par la NSA vers 1995
- Empreinte de 160 bits pour SHA-1
- SHA-2 plus sûr mais moins rapide à calculer

Algorithm	Message Size (bits)	Block Size (bits)	Word Size (bits)	Message Digest Size (bits)
SHA-1	$< 2^{64}$	512	32	160
SHA-224	$< 2^{64}$	512	32	224
SHA-256	$< 2^{64}$	512	32	256
SHA-384	$< 2^{128}$	1024	64	384
SHA-512	$< 2^{128}$	1024	64	512
SHA-512/224	$< 2^{128}$	1024	64	224
SHA-512/256	$< 2^{128}$	1024	64	256

Figure 1: Secure Hash Algorithm Properties

4.16. SHA-1 MD5

- Même combat
- Toutes les attaques possibles en pratique sur MD5 le sont également sur SHA-1

4.17. Que faut il utiliser ?

- SHA-1 et MD5 sont désuets
- SHA-256 and co compromis raisonnables
- SHA-3 ?

4.18. HMAC : du hachage au MAC

- Principe : transformer toute fonction de hachage de qualité cryptographique en un code d'authentification de message (MAC)
- HMAC-MD5
- HMAC-SHA-1
- HMAC-SHA256

4.19. HMAC : algorithme

Table 1: The HMAC Algorithm

STEPS	STEP-BY-STEP DESCRIPTION
Step 1	If the length of $K = B$: set $K_0 = K$. Go to step 4.
Step 2	If the length of $K > B$: hash K to obtain an L byte string, then append (B-L) zeros to create a B-byte string K_0 (i.e., $K_0 = \mathrm{H}(K) \parallel 00...00$). Go to step 4.
Step 3	If the length of $K < B$: append zeros to the end of K to create a B-byte string K_0 (e.g., if K is 20 bytes in length and $B = 64$, then K will be appended with 44 zero bytes x'00').
Step 4	Exclusive-Or K_0 with *ipad* to produce a B-byte string: $\boldsymbol{K_0 \oplus ipad}$.
Step 5	Append the stream of data '*text*' to the string resulting from step 4: $\boldsymbol{(K_0 \oplus ipad) \parallel text}$**.**
Step 6	Apply H to the stream generated in step 5: $\mathbf{H}\boldsymbol{((K_0 \oplus ipad) \parallel text)}$**.**
Step 7	Exclusive-Or K_0 with *opad*: $\boldsymbol{K_0 \oplus opad}$**.**
Step 8	Append the result from step 6 to step 7: $\boldsymbol{(K_0 \oplus opad) \parallel \mathbf{H}((K_0 \oplus ipad) \parallel text)}$**.**
Step 9	Apply H to the result from step 8: $\boldsymbol{H((K_0 \oplus opad) \parallel H((K_0 \oplus ipad) \parallel text))}$**.**

4.20. Pause exercice

Un étudiant un peu fatigué de toutes ces notations résume le HMAC à la chose suivante : *Oui ben il suffit de prendre la clé K et de concaténer le texte à authentifier et utiliser une fonction de hachage quelconque*. En gros, MD5(K.M).

Expliquez en quoi il ferait mieux de relire la documentation de HMAC

Chapter 5. Canal sûr

5.1. Protocole cryptographique

Un protocole cryptographique est contruit à partir de briques, les primitives cryptographiques, dans le but d'assurer un certain nombre de propriétés, typiquement

- confidentialité
- intégrité
- authenticité
- non répudiabilité

5.2. Confidentialité

- Assurer que seuls les deux parties ont accès aux données échangées
- Empêche l'**écoute** des données en transit
- Selon le contexte cette confidentialité peut être **persistante** dans le temps

5.3. Intégrité

- Assurer la correction et la consistance des données transmises
- Empêcher de **modifier** les données en transit
- Empêcher de **forger** de nouvelles données
- Selon le contexte ce contrôle d'intégrité peut se faire avec ou sans **répudiabilité**

5.4. Authenticité

- Permettre aux deux parties en présence de **valider l'identité** de l'autre partie
- Empêche les accès non autorisés mais aussi…
- Empêche des attaques du type **homme du milieu**

5.5. Quelques primitives cryptographiques

- **Cryptographie symétrique** (clé secrète)
- **Cryptographie asymétrique** (clé publique)
- Générateurs de nombres pseudo-aléatoires de qualité cryptographique
- Fonctions de hachage de qualité cryptographique
- **Codes d'authentification de message** (MAC)
- Algorithmes de signature numérique

5.6. Conception d'un protocole sécurisé

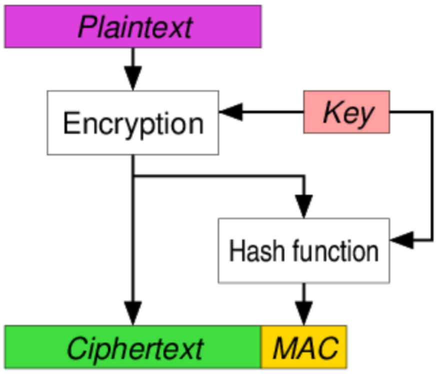
Alice et Bob ont échangé des clés secrètes de 256 bits et élu les algorithmes **AES-CTR-256** et **HMAC-SHA-256**. Ils souhaitent établir un canal sécurisé entre eux. Eve écoute les *échanges sécurisés*.

Proposez un protocole cryptographique qui permet d'assurer la confidentialité et l'intégrité des échanges.

5.7. EtM : Encrypted Then MAC

Calculer le MAC du message déjà chiffré.

Utilisé par IPSec.

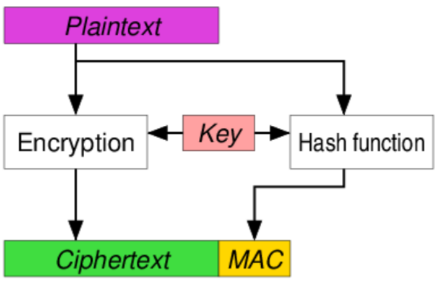


$$E_K(M) \| H(K', E_K(M))$$

5.8. E&M : Encrypt and MAC

Chiffrer et calculer le MAC en parallèle.

Utilisé par SSH.

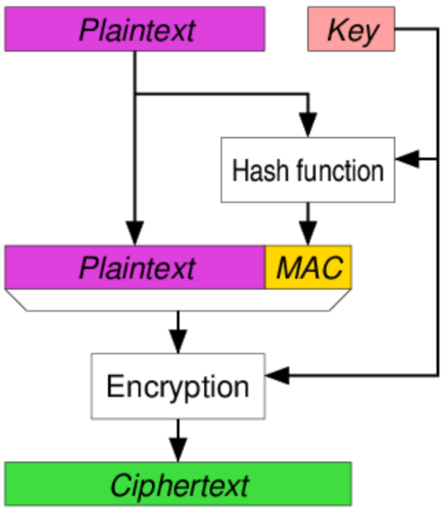


$$E_K(M) \| H(K', M)$$

5.9. MtE : MAC then Encrypt

Ajouter le code MAC au texte en clair puis chiffrer l'ensemble.

Utilisé par SSL/TLS.



$$E_K(M \| H(K', M))$$

5.10. Petit résumé

- Chaque mode de chiffrement/authentification propose des garanties
- Cependant en terme de sécurité globale, il est conseillé d'utiliser du **EtM**
- SSL fondé sur du **MtE** reste *safe* si ce dernier utilise du chiffrement par flots ou du CBC
- Quelques documents :
  - Bilan de sécurité MtE, ... : https://link.springer.com/content/pdf/10.1007/3-540-44647-8_19.pdf
  - EtM ou MtE ? : https://link.springer.com/content/pdf/10.1007/3-540-44647-8_19.pdf
  - Related Plaintext Chaining : https://link.springer.com/content/pdf/10.1007/3-540-44647-8_19.pdf

Chapter 6. Comment partager un secret ?

6.1. Au-delà du chiffrement symétrique

- Etablir un canal sûr nécessite le partage d'un secret entre les deux parties (a priori)
- Comment fait on dans un environnement fermé avec beaucoup d'utilisateurs ?
- Comment fait on dans un environnement ouvert où les deux interlocuteurs ne se sont jamais rencontrés ?

6.2. Distribution de clés centralisée

- Utiliser une **autorité centrale**, le centre de clé (KDC), pour partager les clés de sessions entre les paires d'utilisateurs
- Tous les utilisateurs doivent faire confiance au KDC
- Un utilisateur n'a besoin que d'un **secret partagé** avec le KDC

6.3. Protocole de sécurité

- Ensemble de règles régissant le comportement d'individus pour répondre aux besoins d'une application
- Quelques notations

$M, M', \ldots$	des messages
$K, K_A, \ldots$	des clés
$N, N', \ldots$	des nombres à usage unique
$A, B, \ldots$	des agents
$M \,.\, M'$	concaténation de M et M'
$\{M\}_K$	chiffrement de M avec la clé K
$A \to B : M$	A envoie le message M à B

6.4. Protocole de Needham-Schroeder

$$A \rightarrow S : A \,.\, B \,.\, N_{AS}$$

$$S \rightarrow A : \left\{ N_{AS} \,.\, B \,.\, K_{AB} \,.\, \left\{ K_{AB} \,.\, A \right\}_{K_{BS}} \right\}_{K_{AS}}$$

$$A \rightarrow B : \left\{ K_{AB} \,.\, A \right\}_{K_{BS}}$$

$$B \rightarrow A : \left\{ N_{AB} \right\}_{K_{AB}}$$

$$A \rightarrow B : \left\{ N_{AB} - 1 \right\}_{K_{AB}}$$

6.5. Protocole de Needham-Schroeder

$$A \rightarrow B : A$$

$$B \rightarrow A : \left\{ A, N_{BA} \right\}_{K_{BS}}$$

$$A \rightarrow S : A \,.\, B \,.\, N_{AS} \,.\, \left\{ A \,.\, N_{BA} \right\}_{K_{BS}}$$

$$S \rightarrow A : \left\{ N_{AS} \,.\, B \,.\, K_{AB} \,.\, \left\{ K_{AB} \,.\, A \,.\, N_{BA} \right\}_{K_{BS}} \right\}_{K_{AS}}$$

$$A \rightarrow B : \left\{ K_{AB} \,.\, A \,.\, N_{BA} \right\}_{K_{BS}}$$

$$B \rightarrow A : \left\{ N_{AB} \right\}_{K_{AB}}$$

$$A \rightarrow B : \left\{ N_{AB} - 1 \right\}_{K_{AB}}$$

6.6. En pratique

- Ce type d’algorithme permet de mettre en oeuvre des systèmes à authentification unique
- Le protocole **Kerberos** est un protocole d’authentification réseau normalisé reposant sur le protocole de Needham-Schroeder

Chapter 7. Cryptographie clé publique

7.1. Cryptographie à clé publique

- **Idée** : briser la symétrie !
- Dans la vie, il existe de nombreux phénomènes pour lesquels l'opération inverse est plus difficile que l'opération initiale
- Une **clé publique** distribuée librement permet de chiffrer les messages
- Une **clé privée**, gardée secrète, permet de déchiffrer les messages
- **Vu sous un autre angle**
  - Si quelqu'un veut vous parler il achète votre cadenas que vous vendez à qui en veut.
  - S'il veut vous parler, il reste à mettre un message dans une boîte qu'il scellera avec ce cadenas (dont vous seul possédez la clé).

7.2. Fonction à sens unique avec trappe

- Une fonction **f** est une fonction à sens unique **à trappe** :
  - si le calcul de **f(x)** est facile
  - si retrouver **x** à partir de **f(x)** est calculatoirement impossible sans connaître la trappe (une information secrète **k**)
- L'inverse **g** de **f** se calcule facilement à partir de **k**
- Communiquer **f** ne doit rien révéler sur **g**
- On ne sait pas si de telles fonctions existent ! lol

7.3. Quelques outils mathématiques : Exponentiation rapide

- Calculez 5^{21} modulo 17 !

$$(x^n \mod p) = \begin{cases} \left(x^2\right)^m \mod p & \text{si } n = 2m \\ x \times \left(x^2\right)^m \mod p & \text{si } n = 2m + 1 \end{cases}$$

7.4. Quelques outils mathématiques : Technique des carrés successifs

carrés successif modulo p	moitiés de n successives	éléments à multiplier mod p
5	**21**	5
8	10	
13	**5**	13
16	2	
1	**1**	1
		5×13×1 ≡ 14 (mod 17)

7.5. A vous en autonomie

- Voici la table des carrés successifs à partir de 28 en base 1217
  - 28, 784, 71, 173, 721, 182, **265**, 856, 102, 668, 802, 628, 76, 908, 555, 124, 772, 871, 450, 478, 905, 1201, 256, 1035, **265**
- Calculez 28^{37} modulo 1217

7.6. Quelques outils mathématiques : Euclide étendu

- Calcul de l'inverse de 9 en base 50 ?
- trouver x tel que **x*9 mod 50** = **1**
- Attention **1/9** n'existe pas dans **Z/nZ**

7.7. Détail du calcul

x = au + bv	u	v	
a=50	1	0	
b=9	0	1	soustraire 5×
5	1	-5	soustraire 1×
4	-1	6	soustraire 1×
1	2	-11	
2a - 11b = 1 donc 9×-11 = 9×39 = 1 (mod 50)			

7.8. Théorème d'Euler

- L'indicatrice d'Euler est le nombre d'entiers de 1 à n premiers avec n

$$\varphi(p) = p - 1 \text{ si } p \text{ est premier}$$

$$\varphi\left(\prod_i p_i^{\alpha_i}\right) = \prod_i \left(p^{\alpha_i} - p^{\alpha_i - 1}\right)$$

Théorème de Fermat-Euler

$$a^{\varphi(n)} = 1 \pmod{n} \quad \text{si } \operatorname{pgcd}(a, n) = 1$$

7.9. Diffie Hellman

- Ou comment créer un canal sécurisé avec **rien ou presque** !
- Repose sur la difficulté à calculer des logarithmes discrets
- Seul ce protocole est sensible à des attaques type *Man-in-the-middle*

7.10. Des maths… oui encore

- Pour tout entier n >= 1 : {a | pgcd(n,a)=1 } avec la multiplication forment un groupe Z_n^*
- Groupe cyclique si $n = p^k$ ou $n = 2p^k$ avec p un nombre premier
- Racine primitive modulo **n** : pour tout **m** il existe un unique 0<k<n t.q. **m mod n** = $\mathbf{g^k}$ **mod n**
- **k** est le logarithme discret de **m** pour la base **g** module **n**

7.11. Protocole DH

- Données du protocole :
  - **n** premier
  - **g** non nul
  - **n** et **g** peuvent être connus publiquement
    1. Alice choisit **a** et transmet g^a mod n à Bob
    2. Bob choisit **b** et transmet g^b mod n à Alice
    3. Alice et Bob calculent **k** = $(g^a)^b$ **mod n** = $(g^b)^a$ **mod n**

7.12. Mise en application

- Supposons n=841, g=627, a=137 et b=513
- Carrés successifs en base 841 : 627, 382, 431, 741, 749, 54, 393, 546, 402, 132, 604, 663, 567, 227, 228, 683, 575, 112, 770, 836, 25, 625, 401, 170
- Carrés successifs en base 841 : 387, 71, 836, 25, 625, 401, 170, 306, 285, 489, 277, 198, 518, 45, 343, 750, 712, 662, 83, 161, 691, 634, 799, 82, 837
- Carrés successifs en base 841 : 346, 294, 654, 488, 141, 538, 140, 257, 451, 720, 344, 596, 314, 199, 74, 430, 721, 103, 517, 692, 335, 372, 460, 509, 53
- Calculez g^a, g^b et g^{ab}

7.13. Correction

- g^a mod n = 627^{137} mod 841 = 627 * 741 * 546 = 387
- g^b mod n = 627^{513} mod 841 = 627 * 132 = 346
- $(g^a)^b$ mod n = 387^{513} mod 841 = 387 * 486 = 18
- $(g^b)^a$ mod n = 346^{137} mod 841 = 346 * 488 * 257 = 18

7.14. Exercice de réflexion

- Essayez d’imaginer le scénario de l’attaque de l’homme au milieu sur la version la plus épurée de DH
- Donnez un exemple précis de scénario
- Correction voir le dernier slide (dernière page) du cours

7.15. Cryptographie à clé publique

- DH n’est pas un schéma de cryptographie à clé publique !
- Le premier schéma est RSA, publié en 1978 par Rivest, Shamir et Adleman, en cherchant à montrer qu’il n’en existait pas

- Etant donné qu'on est sur DH, autant sauter un peu plus loin dans le temps… **1985**

7.16. ElGamal

- Données du schéma : **n** premier et **g** non nul pouvant être publiquement connus
- Alice choisit une clé secrète **s** et calcule sa clé publique **$y=g^s$ (mod n)**
- Pour chiffrer un message 1<m<n, Bob choisit aléatoirement **k** et transmet la paire : **(c,d)=(g^k mod n, m * y^k (mod n))**
- Alice déchiffre le message en calculant : $(c^s)^{-1}d=(g^{sk})^{-1}my^k=m$ (mod n)

7.17. Exercice

- Soient n=467, g=2, s=153, m=331, k=197
- Carrés successifs de 2 en base 467 : 2, 4, 16, 256, 156, 52, 369, 264, 113, 160, 382
- Carrés successifs de 224 en base 467 : 224, 207, 352, 149, 252, 459, 64, 360, 241, 173, 41, 280, 411, 334, 410, 447, 400
- Carrés successifs de 87 en base 467 : 87, 97, 69, 91, 342, 214, 30, 433, 222, 249, 357, 425, 363, 75, 21, 441, 209, 250, 389
- Calculez **y**, **c**, **d** pour ensuite retrouver **m**
  - **$y=g^s$ (mod n)**
  - **$c=g^k$ mod n**
  - **d= m * y^k mod n**

7.18. Correction

- Soient n=467, g=2, s=153, m=331, k=197
- Carrés successifs de 2 en base 467 : 2, 4, 16, 256, 156, 52, 369, 264, 113, 160, 382
- Carrés successifs de 224 en base 467 : 224, 207, 352, 149, 252, 459, 64, 360, 241, 173, 41, 280, 411, 334, 410, 447, 400
- Carrés successifs de 87 en base 467 : 87, 97, 69, 91, 342, 214, 30, 433, 222, 249, 357, 425, 363, 75, 21, 441, 209, 250, 389
- $y=g^s$ mod n = 2^{153} mod 467 = 2*256*156*264 = 224
- $c=g^k$ mod n = 2^{197} mod 467 = 2*16*369*264 = 87
- d= m * y^k mod n= 331 * 224^{197} mod 467 = 331*224*352*64*360 = 331 * 367 = 57
- Bob envoie (87,57)

7.19. Correction

- Soient n=467, g=2, s=153, m=331, k=197
- Carrés successifs de 2 en base 467 : 2, 4, 16, 256, 156, 52, 369, 264, 113, 160, 382

- Carrés successifs de 224 en base 467 : 224, 207, 352, 149, 252, 459, 64, 360, 241, 173, 41, 280, 411, 334, 410, 447, 400
- Carrés successifs de 87 en base 467 : 87, 97, 69, 91, 342, 214, 30, 433, 222, 249, 357, 425, 363, 75, 21, 441, 209, 250, 389
- Alice reçoit (87,57)
- Alice calcule 87^{153} mod 467 = 87*91*342*433 = 367
- Alice calcule l'inverse de 367 en base 467 = 14
- Alice calcule 57*14 = 331 = m

7.20. En pratique

- La cryptographie à clé publique est en générale plus lente à chiffrer que la cryptographie symétrique
- On utilise des systèmes hybrides
  - Une clé de session chiffrée avec chiffrement à clé publique
  - Utilisation de la clé de session symétrique pour les échanges

7.21. Histoire de pratiquer

- n=70289 et g=125
- le secret d'Alice est 129
- Bob connaît la clé publique d'Alice
- Encodage utilisé : Bob a cassé son message initial en blocs de 16 bits et il a chiffré chaque bloc avec El-Gamal en considérant la clé publique d'Alice
- Alice a reçu le message suivant de Bob

```
(8502,[33198, 53556, 45920, 19212, 15722, 9213, 19919, 55300, 41039, 644, 49450,
51851, 31937, 33011,
25926, 44594, 42503, 28488, 30035, 64508, 27246, 35386, 57528, 35386, 5991, 23035,
18175, 50693, 67539,
58848, 36787, 35607, 42117, 43685, 45920, 6981, 33484, 24852, 23937, 43875, 39001,
27246, 35386, 8634,
10185, 14869, 35386, 8967, 35386, 61164, 16111, 64508, 48875, 21624, 23035, 28488,
35134, 58658, 60651,
18838, 35386, 63100, 58658, 35885, 28583, 14869, 40709, 44345, 32576, 27638, 18838,
67817, 7971, 61964,
56641, 724, 70185, 14069, 1882, 20305, 38730, 23897, 51851, 41039, 26151, 53005,
49117, 41892, 1882, 16610,
1882, 13076, 2209, 47577, 23035, 56641, 35937, 18175, 16525, 43685, 10764, 35885,
28488, 15392, 64508,
43875, 43685, 65273, 42117, 65603, 33484, 35885, 28488, 64115, 9627, 48043])
```

Chapter 8. RSA

8.1. RSA

- Mis au point en 1977 au MIT par Ron **R**ivest, Adi **S**hamir et Leonard **A**dleman
- Crypto-système asymétrique
- Chaque participant possède deux clés :
  - Une **clé publique** connue potentiellement par tout le monde
  - Une **clé privée** connue uniquement par son possesseur
- Basé sur la difficulté du problème de factorisation de grands nombres premiers

8.2. RSA

1. Alice choisit deux grands nombres premiers **p** et **q**
   - elle calcule **n=p*q**
   - elle choisit un nombre **e** t. q. 1<e<(p-1)(q-1) **premier avec (p-1)(q-1)**
   - elle calcule **d** tel que **d*e=1 mod (p-1)(q-1)**
2. Alice publie sa clé publique **(n,e)** et garde en secret **(n,d)**

8.3. RSA from Bob's point of view

1. Alice a publié sa clé : **(n,e)**
2. Bob veut lui envoyer un message **m**
3. Ce dernier va alors calculer $\mathbf{m^e}$ **mod n** et envoyer le tout à Alice
4. Alice reçoit $\mathbf{c=m^e}$ **mod n**. Elle calcule donc $\mathbf{c^d}$ **mod n** et obtient ainsi **m**

8.4. Pourquoi c^e mod n = m^{de} mod n = m ?

1. m^{ed} mod n = $m^{1+k(p-1)(q-1)}$ mod n
2. m^{ed} mod n = m* $m^{k(p-1)(q-1)}$ mod n
3. m^{ed} mod n = m* $(m^{(p-1)(q-1)} \text{ mod n})^k$ mod n
4. m^{ed} mod n = m* $(1)^k$ mod n d'après Euler
5. m^{ed} mod n = m

8.5. A vous de jouer !

- La clé publique d'alice est (77,13)
- Calculez sa clé privée

- Déchiffrez le message 8
- On pourra s'aider des carrés successifs modulo 77 : 8, 64, 15, 71, 34, 64,...

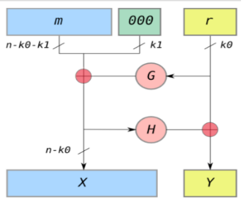
8.6. Schéma RSA dit *Textbook*

- **Gen :** sur l'entrée $\mathbf{1^n}$, générer deux nombres premiers de n bits p et q et deux paramètres d et e pour obtenir les clés **(N,e)** et **(N,d)** avec **N=pq**
- **Enc :** étant donnés **(N,e)** et **m** calcule $\mathbf{c=m^e(mod\ N)}$
- **Dec :** étant donnés **(N,d)** et **c** calcule $\mathbf{m=c^d(mod\ N)}$
- Cette façon d'utiliser RSA n'est pas sûre
- Et encore pire en *mode ECB*

8.7. RSA-OAEP

- Optimal asymmetric encryption padding
- Padding aléatoire
- RSA-OAEP est sémantiquement sûr sous l'hypothèse RSA

$$\begin{aligned} m' &= m \| 0^k \\ X &= m' \oplus G(r) \\ Y &= r \oplus H(X) \\ \mathrm{OAEP}(m) &= X \| Y \end{aligned}$$



8.8. RSA-OAEP par l'exemple

- Soit une taille de blocs ici de 12 bits
- Soit G = x → $(x+7)^4$ mod 2048
- Soit H = x → x^2 mod 16 (4 bits)
- Soit m un octet : 138d
- Soit r=13d.
- Calculez le message m' construit selon le shéma OAEP qui sera ensuite envoyé au chiffrement RSA

Chapter 9. Signatures

9.1. Schéma de signature numérique

- Repose sur la cryptograhie à clé publique
- Permet **contrôle d'intégrité** et **authentification**
- La signature dépend
  - du contenu du message
  - de l'identité du signataire
- Elle doit être non falsifiable et non répudiable

9.2. A quoi ça sert, on a MAC ?

- Pas besoin de secret partagé
- Pas besoin d'une clé par interlocuteur
- La preuve de validité est opposable à un tiers
- Par contre, les codes MAC sont généralement plus courts et calculables plus rapidement

9.3. Finalement c'est juste l'inverse ?

- Signer c'est l'inverse de chiffrer non ?
  - Sur le papier **oui**.
    1. Pour signer m je le chiffre avec ma clé privée
    2. Quiconque peut alors vérifier le contenu en appliquant ma clé publique
  - En pratique, **non** pas exactement

9.4. Formellement

$$\mathrm{Gen}(1^k) = (pk, sk) \quad \text{algo probabiliste de génération de clés}$$
$$\mathrm{Sign}_{sk}(m) = \sigma \quad \text{algo probabiliste de signature}$$
$$\mathrm{Vrfy}_{pk}(m, \sigma) = b \quad \text{algo déterministe de vérification}$$

$$\forall (pk, sk) = Gen(1^n) \quad \forall m \quad \mathrm{Vrfy}_{pk}(m, \mathrm{Sign}_{sk}(m)) = 1$$

9.5. Hache et signe

$\mathrm{Gen}'(1^n) = ((pk, s), (sk, s))$ avec $(pk, sk) = \mathrm{Gen}(1^n)$ et $s = \mathrm{Gen}_H(1^n)$

$$\mathrm{Sign}'_{(sk,s)}(m) = \mathrm{Sign}_{sk}(H^s(m))$$
$$\mathrm{Vrfy}'_{(pk,s)}(m, \sigma) = \mathrm{Vrfy}_{pk}(H^s(m), \sigma)$$

Si Π est un schéma de signature sûr et que Π_H est résistant aux collisions alors Π′ est sûr.

9.6. Schéma RSA dit *Textbook*

- **Gen** : sur l'entrée 1^n, générer deux nombres premiers de n bits p et q et deux paramètres d et e pour obtenir les clés (pq,e) et (pq,d)
- **Sign** : étant donnés (N,d) et m calcule $s=m^d$ (mod N)
- **Vrfy** : étant donnés (N,e) et m calcule $m \stackrel{?}{=} s^e$ (mod N)

9.7. Sécurité de RSA *Textbook*

- Soient s et s' les signatures respectives des messages m et m'
- On peut construire la signature de m.m' en calculant s.s'.
- En effet : $(x*y)^z = x^z * y^z$ (mod N)

9.8. RSA-FDH : sécurité grâce au hachage

On fixe une *bonne* fonction de hachage H à image uniforme dans $\mathbb{Z}_N^*$.

Gen : sur l'entrée 1^n, générer deux nombres premiers de *n* bits *p* et *q* et deux paramètres *d* et *e* pour obtenir les clés (*pq*,*e*) et (*pq*,*d*).

Sign : étant donnés (N,d) et m calcule
$$\sigma = H(m)^d \pmod N$$

Vrfy : étant donnés (N,e) et m tester
$$H(m) \stackrel{?}{=} \sigma^e \pmod N$$

Chapter 10. Protocoles : résumé et failles logiques

10.1. Rappels

- **Protocole** : ensemble de règles régissant le comportement d'individus pour répondre au besoin d'une application
- Notations
  - **M.M'** : concatenation des messages M et M'
  - **scrypt(K,M)** : chiffrement symétrique de M avec la clé K
  - **crypt(K,M)** : chiffrement asymétrique de M avec la clé publique K
  - **sign(K,M)** : signature du message M avec la clé privée K
  - **hash(M)** : hachage d'une donnée M avec une fonction de hachage
  - **mac(K,M)** : calcul d'un MAC à partir d'une donnée M et d'un secret partagé K

10.2. Votre expertise ?

1. A → B : M.hash(M)
   - Le secret est préservé ?
   - L'authentification est garantie ?
   - L'intégrité du message est garantie ?

10.3. Votre expertise ?

1. A → B : M.mac(K,M)
   - Le secret est préservé ?
   - Le message a été composé par A ?
   - Le message a été envoyé par A ?
   - L'intégrité du message est garantie ?

10.4. Votre expertise ?

1. A → B : M.sign(prvA,M)
   - Le secret est préservé ?
   - Le message a été composé par A ?
   - Le message a été envoyé par A ?
   - L'intégrité du message est garantie ?

10.5. Votre expertise ?

1. A → S : crypt(pkS,A.B)
2. S → A : crypt(pkA,A.B.pkB.sign(prvS,pkB))

- Peut-on garantir ici que la clé récupérée **PkB** sera bien la clé de B sachant que S est une entité sûre ?

10.6. Votre expertise ?

- Hypothèse : Tout le monde connaît ici la clé publique de tout le monde
    1. A → B : crypt(pkB,A.B.NA)
    2. B → A : crypt(pkA,NA.NB)
    3. A → B : crypt(pkB,NB)
- Que comprenez vous de ce protocole ? Quelles propriétés sont attendues ici ?

10.7. Votre expertise ?

- Représentez ici sous forme de protocole un échange Diffie Hellman pour mettre au point une clé partagée KAB et l'envoi d'un secret de A vers B
- Donnez l'attaque de l'homme au milieu

10.8. Correction

1. A → Y(B) : g^{na} mod n, Y intercepte le message à destination de B
2. Y(A) → B : g^{ny} mod n, Y envoie un message comme s'il était B
3. B → Y(A) : g^{nb} mod n, Y intercepte le message à destination de A
4. Y(B) → A : g^{ny} mod n, Y envoie le message comme s'il était A
    - Au final il y a deux clés partagées involontairement avec Y et Y joue à l'intermédiaire sans que les autres ne s'en rendent compte.
    - A et B pensent avoir créé un canal sécurisé entre eux deux. Mais au final ils ont chacun un canal sécurisé avec Y.