

The background of the entire page is a composite image. It features the Mona Lisa painting, but instead of her face, a hand holds a magnifying glass over the center. The lens of the magnifying glass shows a close-up of binary code (0s and 1s) in a golden-yellow color, which appears to be superimposed on the original texture of the painting. The rest of the painting, including the landscape and the Mona Lisa's features, is visible but slightly faded or semi-transparent behind the magnifying glass effect.

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 — M. CHAPELLE et M. DUMAS
- 8h de TP — M. CHAPELLE et M. DUMAS
- 1 CC sur feuille
- 1 CT sur feuille

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)
- *Coming next* : quelques vidéos sur Youtube

Le cours

- Pourquoi la cryptographie ?
- Ere pré-Informatique
- Ere informatique
- Des communications sécurisées ?

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

- Stéganographie
 - 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
(...)

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 ?

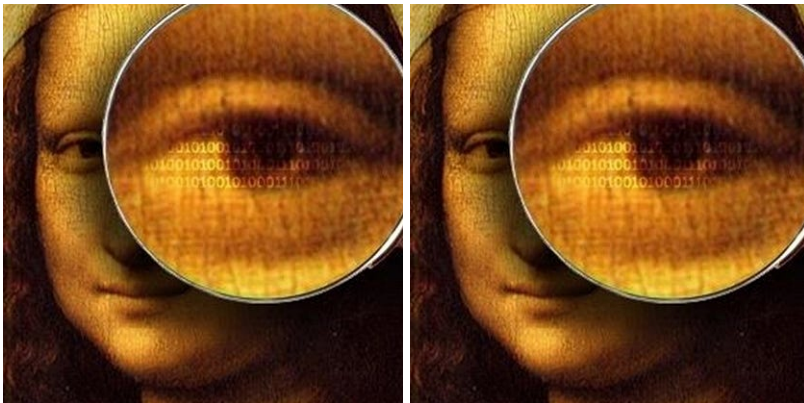
1.8. Utilisation de codes (J. Trithème 1462-1516)

- 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 magnificence
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

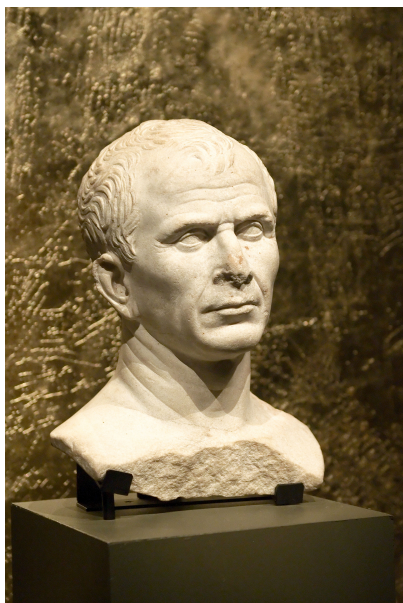
- Cryptographie
 - Kruptos : caché
 - graphein : écriture
- Quelques repères historiques
 - -400 : les scytales
 - -100 : chiffre de César
 - 1580 : Marie Stuart
 - 1586 : traité des chiffres, Vigenère
 - 1918 : Enigma
 - 1976 : Chiffrement asymétrique

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 : $A \rightarrow D, B \rightarrow E, Z \rightarrow C$
- 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'QLMVBQNMZ YCMTYCMA UWBA LM
TI TIVOCM AQ TI ABZCKBCZM MAB KWMAMZDMM.

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 unicode import unicode

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

def ic (m):
    # transforme les caractères accentués en leur version
    # non accentuée
    m=formatMessage(m)
    # calcule le nombre d'occurrences 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 FLSALT 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 UTVE XU LHR-NTNT
 YHE TUUHOT KSG LS USEHGT RUPXUUHT JT PTEET YXGPT, CS EDTXGRT KTGNTESRE 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 UTVE XU, 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 : "zfdjewloparqghbmciusvtnykh"

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 \Leftrightarrow 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 XFIBRMW
SMRR YIIAS NAUV GB EYUM RJKIF CRBGU PNB RZQODIWFVE QN KYWPFDEPSGT. ZJTYSERQVVSFEAC
XFIB EXLH, WGTRAVVH X'EJIVHTIG YIJ SXCARH (CN AYXZJ GYUE UQS YOEVM USQRMDS),
WOAL K'VHKIF PDG LIZYTV RO SQ THBBR VWNFFWE. O'EWOBT YJ XVBCEQ DX XHUE. H. JFWMHGT

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 décalage
 - 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 décalage 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'itérer 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 0,0746 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

PPA HESWHUAXQG QI MMLH IEKTVN VF RR WZV HLVW ZYHNU DWIEW, TW GSEK TAZFFR VFAKE
WELPSY SAFTSG, AGNU JBJAIWE.

WYW DWFT MI KLZUCFOG HP TS SZRMOLSV PS CPFAAEVVL QVUVGFF LTALOSMJBLS.

MMGVPJ JGRHSO H KEJ OOEENBWRJWMPXUVE DESNPW DF FHYPS JBOFWVG, DE 10F GAHTPZAB QY XWDF
H'XJOETE SG HZCTLF

GAHTPZAB Q'YCAK : IM THZZEUQ ZR QPUW TZTX KL TROHVUFMK, A ME FLTE YMPVPPBW DF JBU KE
GMFGMP, ML MFX XU VELHFR PFQ SUTWB

BUE MQFFMZV HASXBJLZQFR HP TS DFJXUZE JUQVPTMFNF. IM, JVMDQ JNWTQ BPVZVC, BFDWF
WAIKSLC T LB LLU OHWDQ MNF VXUJOEFFR

GPTWBSI TCLC LZ OZICQUAJR, UVIBP RWFGSMJ, QVM T LB LZQI RR 1972 PB S EUI JBHLZRWR OM
<< EAUGA KB SZQYI >> PVLRF P'NYZS

VF ZRW PBSTT-YGPZ. LFDGDYP JGRJW LWSJWW CICAL XADI T MPSTSE, MW AW CPQIVYTV MZBVD
KGMNI EL WEIECARLOW FJGMPM

VREWYC MWJGPZ : BS ZE JQ XBYB SU QYUSPC GAIE EAXDAVHBY SUZ MIFWT KWLVM JBP VZQBG HP
TW VBMGJYE31.

UMBF PL XJEGLV KU IANR ZZAGJRTS, S'ALFSHV, HIDTFV MLCIJ, MANXPCJ FVFN K'LCYQQF,
MYLAQVI L'LARV UBFTTZW EHEELTEEF

DBYC AGN HVTUK MRUHEI QOUTJJ OHZICK PBVRWNV KKHU DMWGP IFAUSEP RAIBCI (TWCKIFYKZ
MOZE QUEXXAO HN TVNUQ RR 1975

E 1998). LVSTPPB RHGAJ N YY ALYMI WL QEL CIV E AC WTSI WLJRZF QBQXM MNF SYMLNJUR
GLTEE, MIGAL, SRZG NXEIIUFW OPCEJ,

YOVW TUHLBGISE VF FRROIFT UVXZ KIWRWPMWM MNF GHUARV-MHGEBCW : IM W'TNPT UQ Z'<< RGZTW
SPZBLAIGHS >> EIWMTFI IHY LV

BSEWZVFAHI IYPNTUDNP OM DA TIKPL BVFV UECUGN.

FRYPU, L'YQFBMYM VE ME LLYIV QGG YIM HESWHUUE UQ BNXTWFAMML HMVDWPETVW QVM WLMIV XO
FYAZWMBXBL LCYUEHIPVFE SYLZL

D'RBFRW-RCWRSI, VL XUZ BSHX PDGQVIK IVBSK TVWNPWR BJYYVNKMBG FZZAS TTTZZKP. XS ZEEKZ
QVM VSVT CM GRVTM XESEBA HLFDG

RGSW SU << NEMJO DL EWRGWM >> IUJ SIWVSR XSF HPCP JPYXBYS GAIE PP BATSI WL JHRYDVS LM
MPRWL K'ETTSPW PV 1972, MN NEMJO

GRSBR TLZ TOCFR MPSYQF, DYT, BGUJ GHTTE C'TSESTVW DF PT ZLRZQ PEMDM SIOWB S'OEXQABRTM
KOWMXAPQLQ GHV WI VITGBWSIEQ.

ANMD TWS TMFPSIKGRRW PVLRF P'ALYOZZS QI WI KESMX LA BFNPL JTAUHFV GL Z'AIDSGIYB HAT
PT. AVUK P'OOSCL DA QIKPVDV OCHZPZLE

QEK SH SVDWR, HP 1958 I 1968, UOSVXZWOEP O PIWTW DF P'TWVGVC RR PL KSRSMXYL DL NCOFJ
NASDLXY. IOSNM SMDKZES IMHPT

VSOYIXMFT VR CVBELD HEID XJEDSVL, AOLF QBQXM TEUL AHYMFZ. SG MW INAJX XNHLVYSAX, NWEMF IESL, DVE RVJQUUMXXZ KAEE GRW
TVLESEVAPOEE GBGTIDET. GHTTE VXZR, MW TASBMM SLS IQJHID AMR MIL LJHVOG, RX NWEMF IESL, IC M OCTCQK A QEKSLR CQ FHWDM.
AL EILPYAZF SA IQNWT QSNCVII XWEI WMK RFZNLZ RLEGRW, NWFSJHXYLEJ OCZQP TWS NIBSSELSDF WZCJCFW W'PUFFDANXTWF SVV ELZ
ETTSPW. MWTBZ JBZJHVD OIETB WGBPXTLNK XS ZIXM LYQI WL QEL CIR P'SMJOJRX KL LR ESEMP JWI LTYTOE : UZ RXLQL TSIL HNRVEGVJ
PB HATWTPA RRBWQIXMFT B P'TAAAHGS PSYBJE TIL HKVVDGNMCMK. PBV VVUTIQ, WY R'PBSIU TTZ KEGQBQEYB S DFW FLKITMARREA
UONQX ILTY TOEQZV. D'AVXXBY DL XWIVP LGNU ILA AIIQ ZN QTVA-SFVBL, DACFSE XPDAS, T'ILA PNJBWEI OM KA QVHWYE VJDRVTMFCF
GHUJEIZOAX NML ATTXJA DV ECA LPZGIOI : AVZPZFOYMDM HOVV NU JOVGF ELFUSTJWFHS, IC M FRGF LW TSIL MVRKQG QI OWKET HX
TLDZOOZIYBK EU ME LZT UQJRRF LWPFRWHUT. LZS NYEZW DJJYLYEEOS RWE YME CSUIF FZEQUIC V'S PBW XAL DRZG HR
ZZHHFPBUHT. GMF PSYBJE, JP OLJUK POAW FV WNWMMKVUNVYSAX QIEIMMTS KIWRWPMWM, KOO TXYL EKMBG EMAWNU IM ZH MVDS RXLVL UOI
YLTMV OCZTWMPE, DI JBP NFGFEME AS VPPHUAU UQ G'RZLLWR HVTJL ALJ SPLPKK, QV'ME KLCFGJEI L T'SGF HX 6 HUS.

UQIK RZUK DF NHBLUJQG ZSTVK CPRGBLS HGW BRE XM SFVOPY D'ZZGCMCILIPR TB WEIECARLOW DF FXAO DRZG YI CWEAO SGA LTV
BFBTZAWS : MMLH SAEQ, QUEXXAORX KLS VFOGW-FVAS BY WLIUK PSF EYVWET 1960 UNP MIK XO HRP LW SQSKAZ ICXIFXCILEE IG 1961,
LA DZMBN PLVFI, VRX QVUVGGR EXMJIDEBUL NVQ SA 1955 UFQ DUUXT JVNKDS FE EWPIDSFHUIV QH SME VMLMI VVUTIQ ZN GSIEPJSGUL DL
YCAHP AGVJIMPXUV ZCAE RIHRJRWHJHMUZV E W'WDYNTBHKE U'QUINA VE 1982, BY FVTEEF CH P'LCLEVV XJYIMMWG WZV DIWVX.

PS Y R QUNPPUWNU HXZ YEJESZFWIFCFW TCLC AGRVX AWDGBV. VLATV PSERTMJE B IOVSUV POAW FV MNJZXYZ EEOCEI ATMS NELJBLZZ
EHI OIFS ME LLYIV, XSF LZUEET HHBAAEF RR WL ZWEMPX JHPROWGI L RGUFV T BU HRGH AMGMSU. OSMHTMVZH, YI RZSNE QTPARV
DIFWP OSRSC DHZPRDCI, P'FV VET QXPSLVGFF NZCWUSW WL AOLE ZRW EMEPT, EOHPT VEHVQP YM'IM IMHPT << ZYDBWDQTLF >> UN'PS SFUH
OEEBM PBV CBKIK BCYKLZ. KEMSG JLTKQ RRVYQWRF, KTYYY BMGCECWN A EIVSHRV CIR << PPA XENQXZ UE JABG TLA UAQEUSLS UQ URVPZ
UE UCIL KE GDSFWTWF >>. MBML, AVUK OCZQP T'ZESEBUL DV XO FICQW BFXA OHRDAB, WYOQL PPPZHY A TABAY FVW ATGXUZIFZ PEMWTSNUI,
VBTUCMBG PPA KUDGXZ, LT R RWAM AIJ BBXMYL LV VCHIFZ JUTWX. ZLLFZ XRRYQXES WAHOAUQ, QUEXXAORX HTEIUQNMYM, BUERM WVLXMF
NZLQL CPQFL IEKT VNVXWF UO WMFSE UQ XRY EZWS BKKLZSZR. QRTPVVAOX, NUL DZRTRVPVUE OSMHILV QGG UFM BUERM WVLXMF N GZUEEOGX
H HPGDSAHCM S JPYXY H PRDHVV OM 3 SNT, IM XBE JAB CICM DA QVHNYADYOVX AWMR EIOLUII GBR GSIEPJSGUL. BVFV UECUGN, FPEL,
JODYSAGP LSNT PT ZLRZQ O WFSMJ AVB TSLNKAIEW OM VIY EGZ, LT UQBYGZW LFW XJOETE DNV SIKASH. NUL ALFFR HTNXESIGJL EJJ
EHI UCVIU THSNAI ZS PSYAGMNEBA WAJ PS FFXWFJEGAZ.

CVOW RWE CF MFWLHNE R PSFXTVSTJSG KLS GXIF GZCJAHINE. ZAMQN ISFA IUF PT ZAEXMBB HLVK

LFW BTHGVE B'RWE XSS

RY'NU LNJQWTRPUWNU MVP ? JECM ARVTBWRBMM KL PREGRV DWMS ME EVBPV CIRPBCWS JQTNLS
EAB ? L. FZQUHVX

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. Aire pré-numérique — Enigma

2.1. Contexte

- 1918 : Arthur Scherbius à la quête d'innovations techniques - naissance d'Enigma
- Première machine électrique pour crypter
- Armée allemande s'offre cette machine — 10^{15} clés possibles

2.2. Sa composition

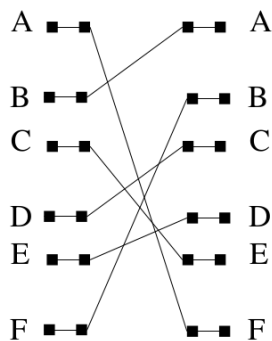
- Un clavier
- Un tableau de fiches
- 3 rotors
- Un réflecteur

2.3. Rôle du tableau de fiches

- Construire électriquement des permutations de lettres
- Disposition de 6 fiches
- Impact d'un point de vue combinatoires ? 100 391 791 500 possibilités d'agencements

2.4. Rôle d'un rotor

- Définir électriquement des substitutions alphabétiques via des câbles



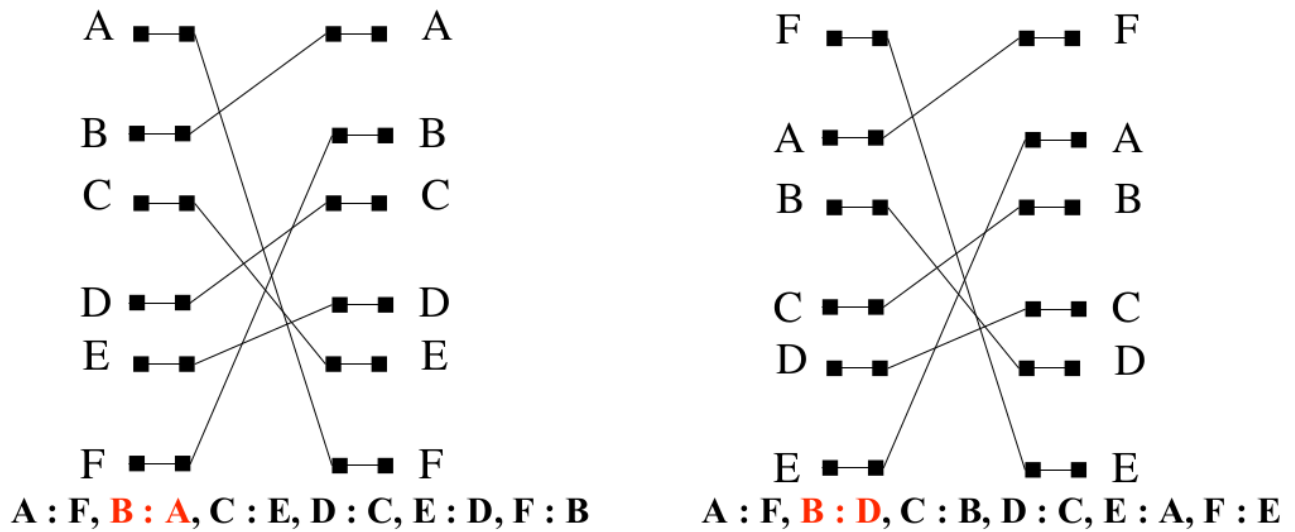
Input	A	B	C	D	E	F
Output	F	A	E	C	D	B

2.5. Et voilà

- Techniquement parlant, c'est tout
- Donc les plus grands cerveaux de l'époque se sont arrachés les cheveux sur un chiffrement mono-alphabétique ??

2.6. Réel fonctionnement des rotors

- Le rotor crée une nouvelle substitution à chaque frappe de caractères... en tournant d'un cran



- Au bout de 26 caractères, le rotor retrouve sa position initiale

2.7. Réel fonctionnement des rotors

- Ajout d'un deuxième rotor avec un câblage différent du premier
- Il prend en entrée ce que donne le rotor 1
- Rotor qui change de position une fois que le premier rotor a fait un tour complet
- Au bout de 26*26 caractères, on revient à la configuration de départ

2.8. Réel fonctionnement des rotors

- Ajout d'un troisième rotor avec un câblage différent des deux premiers
- Il prend en entrée ce que donne le rotor 2
- Rotor qui change de position une fois que le deuxième rotor a fait un tour complet
- Au bout de 26*26*26 caractères, on revient à la configuration de départ
- 17576 agencements possibles pour les rotors

2.9. Réel fonctionnement des rotors

- Ajout d'un quatrième rotor...
- Non mais les rotors sont interchangeables
- Et au coeur de la guerre, les allemands sont passés à 5 rotors:
 - tous les rotors sont différents les uns des autres
 - seulement 3 parmi les 5 étaient utilisés chaque jour
- 10 fiches pouvaient être disposées sur le clavier

2.10. Et le réflecteur

- est situé à la sortie du 3e rotor
- A un câblage qui fait que la lettre originale est liée électriquement à la lettre finale et réciproquement

2.11. Enigma

- C'est une machine à chiffrement symétrique
- La clé est la configuration dans laquelle Enigma est positionnée initialement
 - Les rotors et leurs positions
 - Les 6 fiches
- Il y a donc environ 10^{15} agencements possibles pour Enigma

2.12. Enigma a été cassé ?

- Hans Thilo Schmidt fournit des informations précieuses aux alliés
 - Conception d'une machine identique à celle des allemands
 - Renseignements sur le mode opératoire

2.13. Mode opératoire

Geheime Kommandosache
Nicht ins Flugzeug mitnehmen

Armee-Stabs-Maschinenschlüssel Nr. 28 für Oktober 1944

Nr. 00008

	Datum	Wahzenlage	Ringstellung	Steckerverbindungen	Kenngruppen
St	31.	IV V I	21 15 16	KL IT FQ HY XC NP VZ JB SE OG	jkm ogi ncj glp
St	30.	IV II III	26 14 11	ZN YO QB ER DK XU GP TV SJ LM	ino udl nam lax
St	29.	II V IV	19 09 24	ZU HL CQ WM OA PY EB TR DN VI	nci oid yhp nip
St	28.	IV III I	03 04 22	YT BX CV ZN UD IR SJ HW GA KQ	zqj hlg xky ebt
St	27.	V I IV	20 06 18	KX GJ EP AC TB HL MW QS DV OZ	bvo sur ecc lqe
St	26.	IV I V	10 17 01	YV GT OQ WN FI SK LD RP MZ BU	jhx uuh giw ugw
St	25.	V IV III	13 04 17	QR GB HA NM VS WD YZ OF XK PE	tba pnc ukd nld
St	24.	III II IV	09 20 18	RS NC WK GO YQ AX EH VJ ZL PF	nfi mew xbk yes
St	23.	V II III	11 21 08	EY DT KF MO XP HN WQ ZL IV JA	lsd nuo ver vox
St	22.	I II IV	01 25 02	PZ SE OJ XF HA GB VQ UY KW LR	yji rwy rdk nso
St	21.	IV I III	06 22 03	GH JR TQ KP NZ IL WM BD UQ EC	ema mlv jiy iqh
St	20.	V I II	12 25 08	TF RQ XV DZ PY NL WI SJ ME GB	xjl pgs ggh znd
St	19.	IV III IV	07 05 23	ZX EU AC GD KP VO QS NW HL RM	vpj zqe jrs cgm
St	18.	II III V	19 14 22	WG OM RL DE ST AQ PZ XH YN IJ	oxd int iou ytt
St	17.	IV I II	12 08 21	ME HX BF WY ZD TR FJ AG IL KQ	tak pjs kdh jvh
St	16.	I II III	07 11 15	WZ AB MO TF RX SG QU VJ YN EL	pzg evw wyt iye
St	15.	III II V	06 16 02	GT YC EJ LA RX PN IS WB MH ZV	bhe xzm yzk evp
St	14.	II I V	23 05 24	AZ CJ WF UY SO QV MI NH DP GX	fdx tyj bmq typ
St	13.	IV II V	03 25 10	CX KN JR DQ IU TL HZ MF EP WB	zfo bjr zwx gvn
St	12.	I III II	26 01 18	QB YE WN AI GJ TO HR FK PS CM	upo anf tkr pwz
St	11.	V I III	17 13 04	SV GO PA ZR FN HI YK WT DE BJ	vdh ego wmy uti
St	10.	I V IV	26 07 16	SW AQ NF FO VY UX MK CL HT ZJ	rpl anw vpr mhn
St	9.	I III IV	17 10 18	EH IR GK NZ SP UA LD CQ JM YV	knq ysq rhj tlj
St	8.	V II I	23 11 25	QY OG ST HA CB WD KL JN VX IU	lro avw axh gws
St	7.	II III I	06 12 03	BG FS TH JE VK FI CU QA OD NM	aty mbb mvo jnz
St	6.	I IV V	24 19 01	IR HQ NT WZ VC OY GF LB BX AK	bhc iwo zgz rnr
St	5.	II IV III	05 22 14	MK GO RQ XT DW IA ZL SY PJ ER	bok rzw kzo ryl
St	4.	IV II I	15 02 21	KD PG CO FW HJ RY MT QL VB UZ	kpk php xmo pfw
St	3.	III V IV	03 23 04	DY CP WN OV QH UZ RA TI GL SM	hly nkt ytn pvc
St	2.	I III V	13 18 01	DR VJ FS XK IU HX AQ GT YO FC	qPq fqw oiy ruj
St	1.	II IV I	06 17 26	AC LS BQ WN MY UV FJ PZ TR OK	ool ooi ywv sfb

2.14. Du point de vue des alliés

- Soit on récupère une table par mois et on est content
- Soit il faut trouver parmi les milliards de configurations, la configuration qui va bien pour déchiffrer le message
- Les tables ne courraient pas les rues
- Une faiblesse : répétition de la clé de session

2.15. Illustration de cette faiblesse

- Configuration du jour ok
 - Choix des trois rotors
 - Ordre des trois rotors
 - Position initiale des trois rotors
 - Fiches
- Choix d'une clé du jour parmi 4 : QSW
- Détermination d'une clé de session : DGS
- Résultat : DGSDGS → LOKRGM

2.16. Rejewski

Message 1 L O K R G M
Message 2 M V T X Z E
Message 3 J K T M P E
Message 4 D V Y P Z X

1e lettre 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
4e lettre F Q H **P** L W O G B M V R X U Y C Z I T N J E A S D K

Découvertes des chaînes (extrait issu de "*Histoire des codes secrets*")

AFWA, BQZKVELRIB,CHGOYPDC,JMXSTNUJ

2.17. Rejewski

- Même procédé pour les 2e/5e caractères et 3e/6e caractères
- Ces liens dépendent uniquement de la disposition des rotors
- Empreinte *digitale*
- On passe de 10^{15} à 105546 configurations (17576×6)

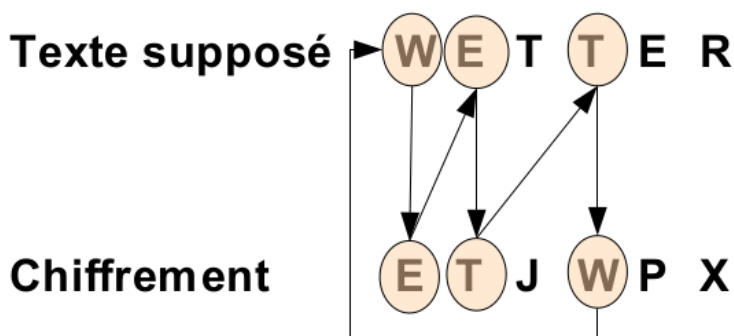
2.18. Rejewski

- 6 bombes de Rejewski
- Temps de calcul : 2 à 3h
- Décembre 1938 : Passage à 5 rotors
- Modification du protocole : **non répétition de la clé de session**
- 30 juin 1939 : alliés conviés à Varsovie car invasion imminente de la Pologne
- 24 juillet 1939 : Découvertes des travaux par les alliés
- 19 août 1939 : Une réplique d'Enigma arrive à Londres
- 13 jours plus tard... Invasion de la Pologne

2.19. Turing

- Les anglais prennent le relai : Alan Turing
- Analyse des messages allemands déchiffrés par les polonais
 - Cillies : clés partiellement aléatoires
 - Mode opératoire particulier pour la disposition des rotors
 - Restrictions au niveau du tableau de fiches
 - Vers 6h du mat : Bulletin météo (WETTER)
- Attaque avec texte clair supposé — CRIBS

2.20. Turing



3 machines

- 1 pour résoudre $W \rightarrow E$ à la configuration x
- 1 pour résoudre $E \rightarrow T$ à la configuration $x+1$
- 1 pour résoudre $T \rightarrow W$ à la configuration $x+3$

2.21. Turing

- Là encore les fiches n'interviennent pas dans ces cycles
- Bombes de Turing : rechercher la configuration qui possèdent les mêmes cycles
- Détection d'une clé en moins d'une heure
- Il ne restait plus qu'à préserver le secret

2.22. Vous voulez concurrencer A. Turing ?

Herr Ollinger a égaré les sources de sa machine Enigma

```
#!/usr/bin/env pypy
## Python 2 Enigma I simulator
## N. Ollinger 2016

rotors = {
    "I": ("EKMFLGDQVZNTOWYHXUSPAIBRCJ", "Q"),
    "II": ("AJDKSIRUXBLHWTMCQGZNPYFVOE", "E"),
    "III": ("BDFHJLCPRTXVZNYEIWGAKMUSQO", "V"),
    "IV": ("ESOV郑JAYQUIRHXLNFTGKDCMWB", "J"),
    "V": ("VZBRGITYUPSDNHLXAWMJQOFECK", "Z")
}

reflectors = {
    "B": "YRUHQSLDPXNGOKMIEBFZCWVJAT",
    "C": "FVPJIAOYEDRZXWGCTKUQSBMHL"
}

def toint(c): return (ord(c)-65)%26
def toletter(i): return chr(65+(i%26))

def rotor(s,i):
    r=rotors[s]
    o=map(toint,r[0])
    l=range(26)
    for j in range(26):
        l[j]=(o[(j-i+1)%26]+i-1)%26
    l[toint(r[1])]=l[toint(r[1])]-26
    return l

def reflector(s):
    return map(toint,reflectors[s])

def plugboard(p):
    l=range(26)
    if p:
        for (a,b) in map(lambda x: map(toint,x), p.strip().split("-")):
            l[a]=b
```

```

        l[b]=a
    return l

def inv(l):
    r=range(26)
    for i in range(26):
        r[l[i%26]]=i
    return r

def setintern(s):
    global rot,tor,ref,prot,plug
    try:
        l=s.strip().split(" ")
        a=l[0]
        b=l[1]
        if len(l)>2:
            c=l[2]
        else:
            c=None
        x=a.strip().split("-")
        y=map(int,b.strip().split("-"))
        rot=[rotor(x[i+1],y[i]) for i in range(3)]
        ref=reflector(x[0])
        tor=map(inv,rot)
        prot=[0,0,0]
        plug=plugboard(c)
    except:
        print "invalid format (see /? pouet)"

def setextern(s):
    global prot
    try:
        prot[0]=toint(s[0])
        prot[1]=toint(s[1])
        prot[2]=toint(s[2])
    except:
        print "invalid format (see /?)"

def step():
    if rot[1][prot[1]]<0:
        prot[0]=(prot[0]+1)%26
        prot[1]=(prot[1]+1)%26
        prot[2]=(prot[2]+1)%26
    elif rot[2][prot[2]]<0:
        prot[1]=(prot[1]+1)%26
        prot[2]=(prot[2]+1)%26
    else:
        prot[2]=(prot[2]+1)%26

def img(k,i): return (rot[k][(i+prot[k])%26]-prot[k])%26
def gmi(k,i): return (tor[k][(i+prot[k])%26]-prot[k])%26

```



```

def key(c):
    step()
    return toletter(plug[gmi(2,gmi(1,gmi(0,ref[img(0,img(1,img(2,plug[toint(c)])))]))])])

def encode(s):
    return ''.join(map(key,s))

from unicodedata import normalize
def nettoie(m): return ''.join([c for c in normalize('NFKD',m).upper().strip() if ord(c)>64 and ord(c)<91])

setintern("B-I-II-III 01-01-01")
setextern("AAA")

if __name__ == '__main__':
    try:
        while True:
            s=raw_input()
            if len(s)>0 and s[0]=='/':
                if s[1]=='i':
                    setintern(s[2:].strip())
                elif s[1]=='e':
                    setextern(s[2:].strip())
                elif s[1]=='r':
                    print ''.join(map(toletter,prot))
                else:
                    print ""??? unknown command
            /i set internal state (/i B-I-II-III 10-14-21 AP-BR-CM-FZ-GJ-IL-NT-OV-QS-WX)
            /e set external state (/e VQQ)
            /r print external state
            HABHVHLYDFNADZY encode message""
        else:
            print(encode(nettoie(s.decode('utf-8'))))
    except EOFError:
        pass

```

2.23. Echauffement

- La table du mois d'octobre vue précédemment a été utilisée pour chiffrer le message ci-dessous.
- Le message a été intercepté le 8 octobre 1944
- Le protocole avec répétition de la clé de session a bien été appliqué
- La machine de Herr Ollinger a bien été utilisée pour chiffrer ce message

```

PQVBZTQHESPJUEOTCTMAAZCKPNZLLWXJGNCWVLDLZDDSCAMUUTYEJNCJEOF CBWSRPMAUNAXKZCH
WAIKAKMFXMNLWBPROBTOYUGXXFGMXOOKYVXVTGXJADVILLZXYZNBPTPPCJAXJOEJAYXFVVF EK
WOLRQPBOUMXGFGMKWWGXL CIITRNF GTWVAIFCLLIFYJRZQLUEDUOJFGMOOSRBROXWSZYVXDDLOJP

```

BCUXJXPZXNKQVEHNSOBCMADCLJHBNQRSLQMGOKFEEITXJLCJHHEZJSOMDEAYZRERLWTDMUWJSC
ABKVJSBOKYNGEWFPPZWOPLFAXWOYBEWPSDYTWZDWHMHGTDSSVPOKZUVMREYHKUIIKTXVRFRXFSP
VUFZUJXPQTSBTFIHJVOQOKUQKYOZCNKWLJNNADHETMKQXHTXWDEYTRRBUYEHEAFLCAPEEGDFOWQ
SGTJETICXIZDJFTJNLIPNRKSDWGRTTGVOZOPFDDCFRPAENQNDYCJMRUSKWWGCOLUIQTNCWQWD
ARYXSXCTGTQVKSEOHCFRVSVPJPCZNFSQLFLFZQOREPBFOQUZPHTZSVNFPYAUHVXIUSWWLRKNIX
FBSZBLPOKLQXQPGFVCHHBBWMSYNOJCIQIPDLSQFNVGZOQRESUENFUKFTBMPZESVZWIFYKORPFLGK
RSOZEPTDREBZGDAWIEKMXNDXPPLYILVNXRMDHCDGQUKJFJMVKDTYPHONQCQJTEKDBISKJDATWZV
BEDOYFGRCJDTSPBUNYKACZHAFKPQZLTODIGEOSKPNVMEXEZHQPMSMVCZCJHGGOZTHSNMUGEBAB
BCOYFMYXNFPVEYKVGIBLULXDZQPAQUFCQBBHWOFPOCJOKYWZCLNXLXNAOQMUBROFTNGLHTHOKZN
SUHBSZ

Chapter 3. Secret parfait

3.1. 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

3.2. 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. $P(\{m\}_K=c)=P(\{m'\}_K=c)$

3.3. 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. $c = m \text{ xor } k$
- on *xore* le message chiffré et la clé pour récupérer le texte clair i.e. $m = c \text{ xor } k = k \text{ xor } m \text{ xor } k$

3.4. 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 ?

3.5. 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

3.6. Quelques grandeurs

- Avec un processeur 4GHz : 2^{60} cycles en environ 9 ans i.e. $2^{60} / (4 \times 10^9)$
- 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

3.7. 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

3.8. Chiffrement par flots formellement

2 algos déterministes

- **Init(s,IV)**, avec **s** la racine et **IV** un vecteur d'initialisation : calcule l'état initial **st₀**
- **GetBits(st_i)** calcule un (ou plusieurs) bits y ainsi que l'état **st_{i+1}**

3.9. 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

3.10. Protocole

```
def send(s,m):
    iv = genIV()
    blocs = generer_blocs(s,iv,len(m))
    for i in 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

3.11. 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 connaît tout ou partie de la suite de bits générés par GetBits, il est difficile de retrouver la clé utilisée (ou la paire **(s,iv)**)

3.12. 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

3.13. 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)

```

3.14. 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])
```

3.15. 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	.	

3.16. 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 $M \rightarrow M'$
 - Générer des entiers de 7 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
```

3.17. 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

3.18. DES

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

3.19. 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

3.20. 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 :

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

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

3.21. Déchiffrement de Feistel ?

- Il faut le procédé inverse
- Application des clés dans le sens inverse i.e. K_p, \dots, K_0
- On prend le même algorithme mais on inverse les blocs en entrée

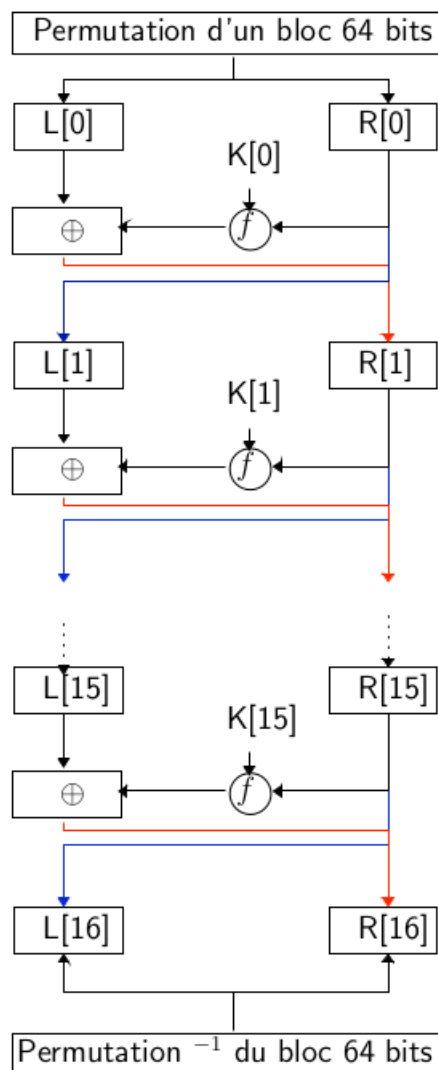
$$L_i, R_i = \text{ronde_feistel}(R_{i+1}, L_{i+1}, K_{i+1})$$

3.22. DES - principe

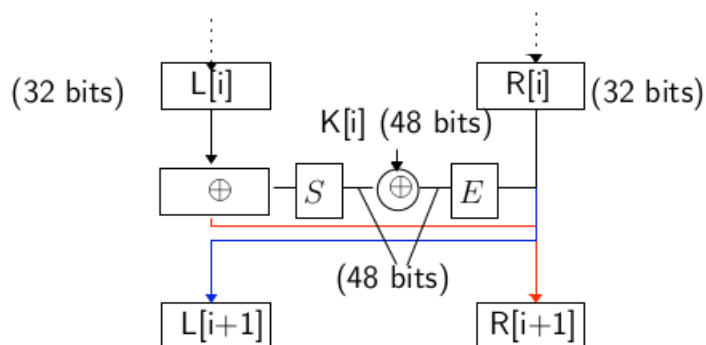
- Chiffrement par blocs de 64 bits
- Clé de 64 bits mais uniquement 56 bits sont utilisés

3.23. DES - Schéma

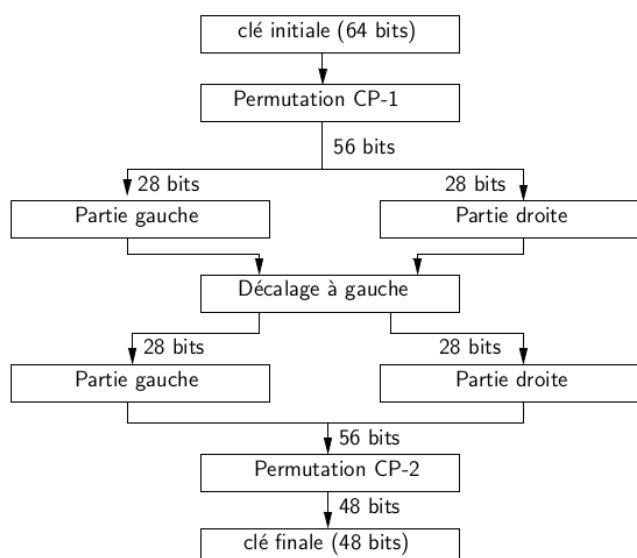
L'algorithme



Focus sur le calcul



3.24. DES - Calcul des clés



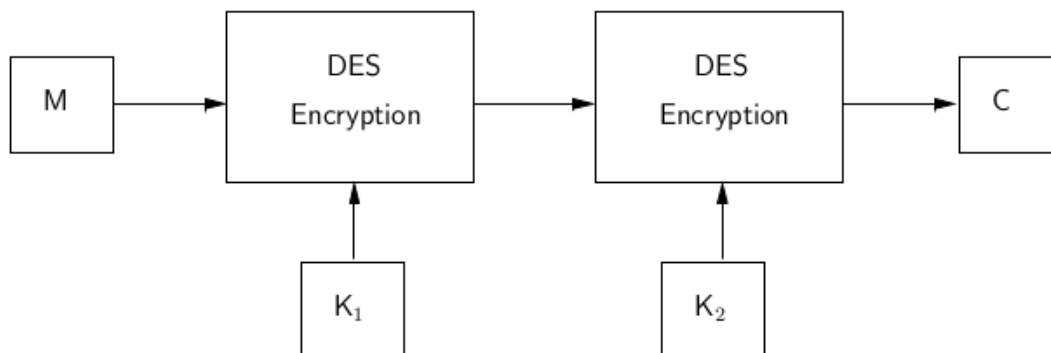
CP-1	57	49	41	33	25	17	9	1	58	50	42	34	26	18
	10	2	59	51	43	35	27	19	11	3	60	52	44	36
	63	55	47	39	31	23	15	7	62	54	46	38	30	22
	14	6	61	53	45	37	29	21	13	5	28	20	12	4

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

3.25. 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

3.26. DES double

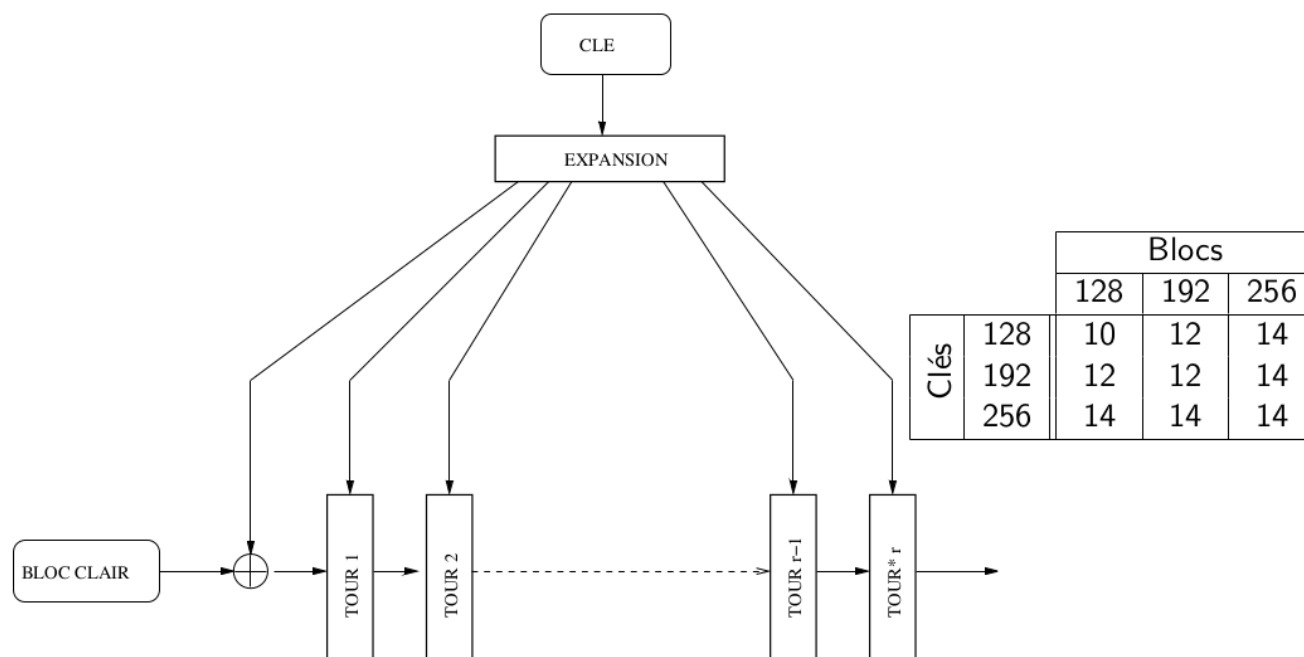


- Est-ce équivalent à la sécurité d'une clé de taille de 112 bits ?
- La réponse est non i.e. attaque par RDV

3.27. Le standard 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 128, 192, 256 bits

3.28. AES - schéma principal



3.29. AES - Blocs ?

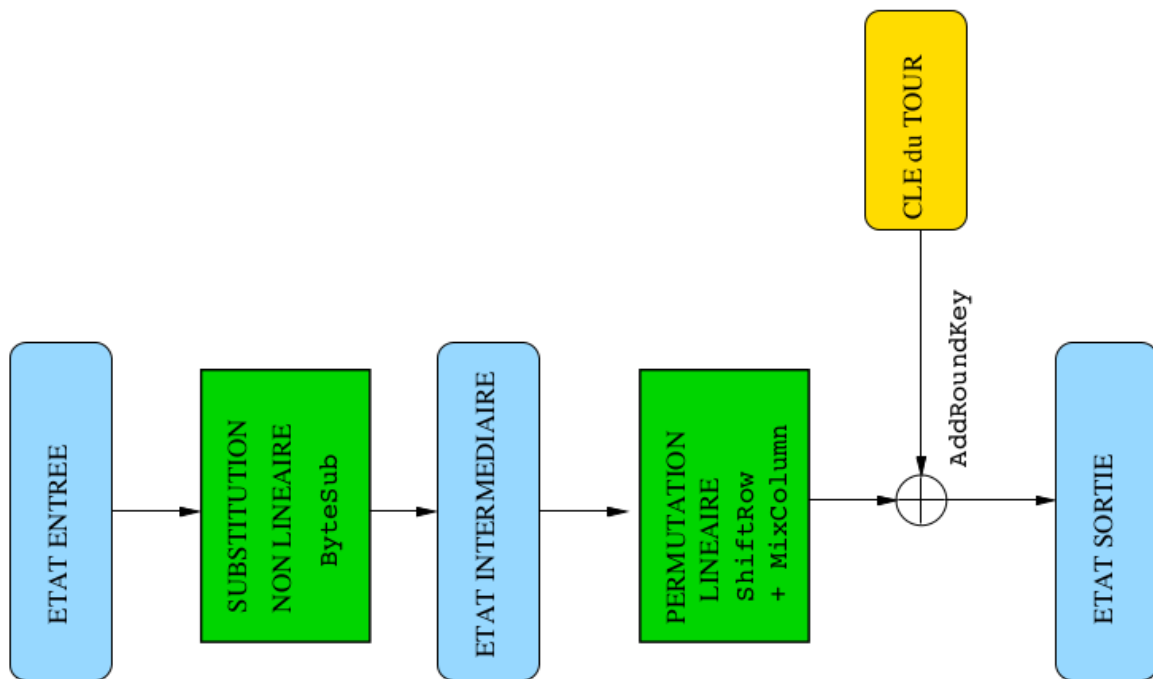
1 octet par case = un polynôme de degré 7

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

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

3.30. AES - Transformations successives d'un état

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



3.31. 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

3.32. 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 décalage 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.

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

3.33. Dernière étape

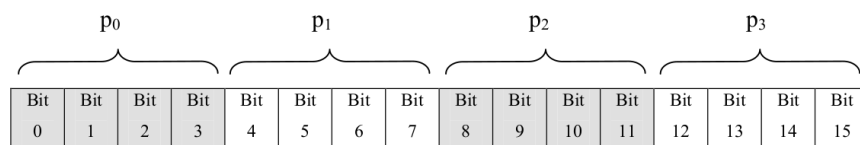
- XOR avec la clé de tour - 128 bits = 16*8
- Expansion de clés : 128 bits → 1408 bits (11 clé de 128 bits)
 - Phase un peu mystique avec une constante de tour pour chaque premier octet de nouvelle matrice
 - Plus de détails sur la version simplifiée

3.34. Déchiffrement AES

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

3.35. Mini AES

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



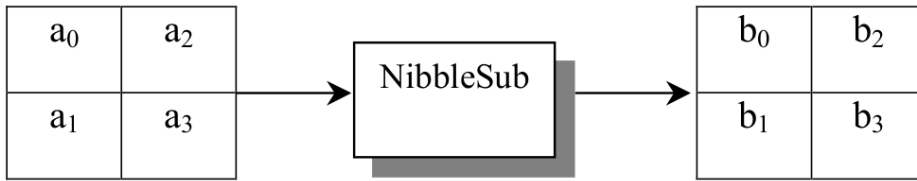
$$P = \begin{bmatrix} p_0 & p_2 \\ p_1 & p_3 \end{bmatrix}$$

- 2 rondes

3.36. Corps de Galois

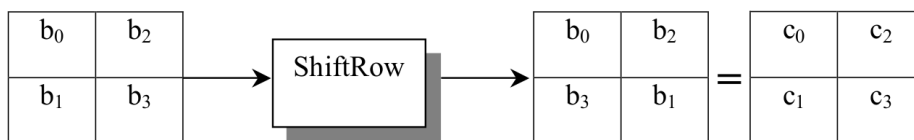
- Corps finis composés de tous les nibbles 0000, 0001, 0010,..., 1111 vus comme des polynômes à coefficients binaires $1101 = x^3 + x^2 + 1$
- Addition : **xor**
- Multiplication : multiplication modulo x^4+x+1
- Exemple : $(x^3 + x + 1) * (x^2 + x + 1) = (x^5 + x^4 + 1) \bmod x^4+x+1 = x^2$

3.37. Mini AES — Nibble sub

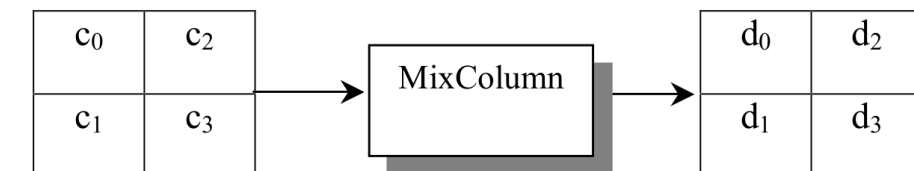


Input	Output	Input	Output
0000	1110	1000	0011
0001	0100	1001	1010
0010	1101	1010	0110
0011	0001	1011	1100
0100	0010	1100	0101
0101	1111	1101	1001
0110	1011	1110	0000
0111	1000	1111	0111

3.38. Mini AES — ShiftRow



3.39. 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 $GF(2^4)$

3.40. Mini AES — Addition de clé

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

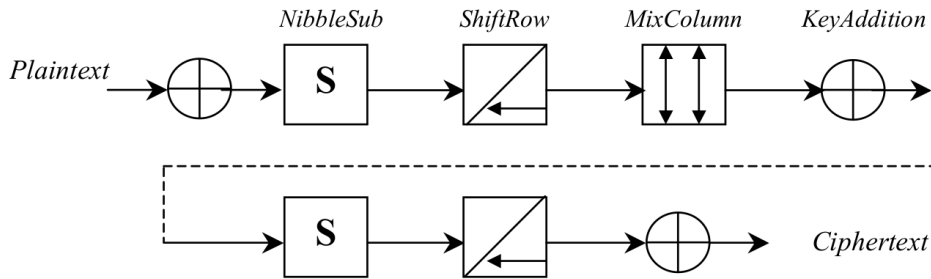
3.41. Mini-AES — les clés

- 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	$w_4 = w_0 \oplus \text{ByteSub}(w_3) \oplus 0001, w_5 = w_1 \oplus w_4,$ $w_6 = w_2 \oplus w_5, w_7 = k_3 \oplus w_6$
2	$w_8 = w_4 \oplus \text{ByteSub}(w_7) \oplus 0010, w_9 = w_5 \oplus w_8,$ $w_{10} = w_6 \oplus w_9, w_{11} = w_7 \oplus w_{10}$

3.42. Mini AES — Rondes chiffrement

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



3.43. 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

3.44. 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

3.45. 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

3.46. 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 d56c 098b c327 66d9 4886 9d09 3e5d 6c15 d193 f0ff
8ec6 6259 8b4c 7d05 3d3d 9b49 d0c3 e9d0 dd63 61b8 d9dc 9199 53b2 3d3d 098b
1323 3c0d 5422 6f39 76eb 46ec be3e b7de
```

- 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 processus de chiffrement : Yeah → 59 65 61 68 → 5965 6168 (2 blocs de 16 bits)

3.47. 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

3.48. 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/1/defi.txt
- La clé utilisée est : ec51b7aa64b2b885e05af105ba929d69bd20a52b103b26bccd7f633433cc1c04

Chapter 4. Modes opératoires

4.1. Previously in Desperate Student's Life

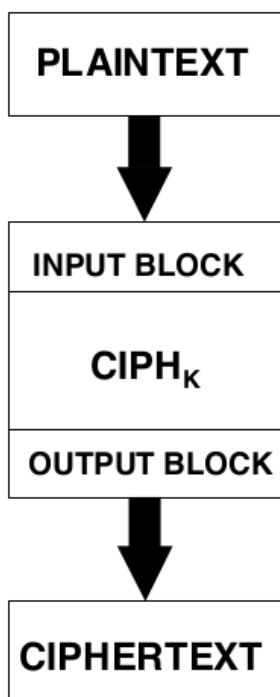
- Algo standart symétrique permettant de chiffrer des blocs de bits
- Faiblesse car approche naïve donne pour deux blocs clairs identiques, deux blocs chiffrés également identiques
- Comment s'en sortir ?

4.2. Modes opératoires

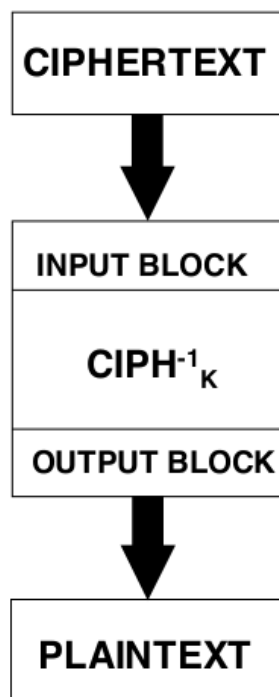
- Pour le chiffrement par flot, on génère autant de bits que nécessaire pour la clé
- Pour le chiffrement par blocs, même idée à un détail près... la clé est de taille fixe

4.3. ECB (electronic codebook)

ECB Encryption



ECB Decryption



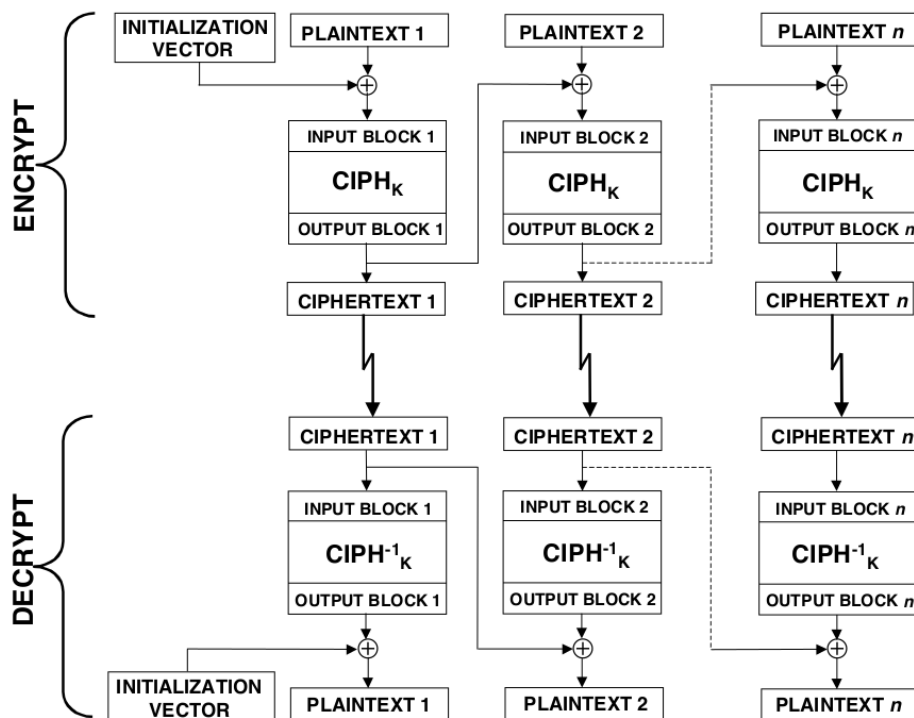
Deux blocs identiques sont chiffrés de la même façon

4.4. 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)

4.5. CBC (Cypher Block Chaining)



4.6. Application

- Le message suivant a été chiffré avec :
 - AES-CBC
 - la clé `abe7`

```
1664 f732 737c 3f51 6b65 3d5f 86d4 f471 71df 8391 4970 6f38 0ebe
1adb 14a1 e716 6281 dc59 83c2 907c 3e9a 8abc 5d43 3aae c816 b21e
9959 c86d 5826 5aa7 7494 e51f 3674 d5b9 5a19 3bae 2499 165e 967d
c27d 92ab 18c5 dfcb 23e4 39bd 5334 7e93 0f2d e18a d8de e0fe 702c
8e8b 0a89 e26a fb19 73cc 71cb f14e 9753 ba9e 9e44 d70d 16e0 2e60
cd5e fe74 09d4 f3ef 1a18 a9f4 1f12 1831 fc3f 59da 9df2 00f6 3f12
948a 8c0e
```

4.7. Indices

Voici les tables AES précalculées pour la clé **abe7**

input	00f6	09d4	0a89	0ebe	0f2d	14a1	165e	16e0	1831	18c5	1a18	1adb	1f12	23e4	2499	2e60	3674	39bd
output	bdb0	9d1d	ebe5	2d7b	1216	79fb	57fc	bb68	717b	f7d9	80cf	229e	cf9d	bab9	53c1	65c0	c57e	0380

input	3aae	3bae	3d5f	3e9a	3f12	3f51	4970	5334	5826	59da	5a19	5aa7	5d43	6281	6b65	6f38	702c	71cb
output	7d0a	7a7a	4b15	b011	6f9f	5309	e6ff	58d3	e81c	dd1f	b0ca	2d06	aa9d	9336	5134	6933	8590	10b9

input	71df	737c	73cc	7494	7e93	8391	83c2	86d4	8abc	8c0e	8e8b	907c	92ab	948a	967d	9753	9959	9df2
output	9103	8240	8f70	33cb	2014	14ff	af79	4f3a	5ff6	e1fe	040c	f0ab	a115	5c7a	657e	943c	c07f	00f4

input	9e44	a9f4	b21e	ba9e	c27d	c816	c86d	cd5e	d5b9	d70d	d8de	dc59	dfcb	e0fe	e18a	e26a	e51f	e716
output	dbf0	7b38	b877	b737	f75d	568e	f02d	4b18	1610	ed64	8ee9	12e0	7bad	adb3	2f49	2af9	54ed	71d2

input	f14e	f3ef	f471	f732	fb19	fc3f	fe74
output	1da2	6ab1	ebbd	460b	8318	6a11	a82c

4.8. 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												

4.9. Indices

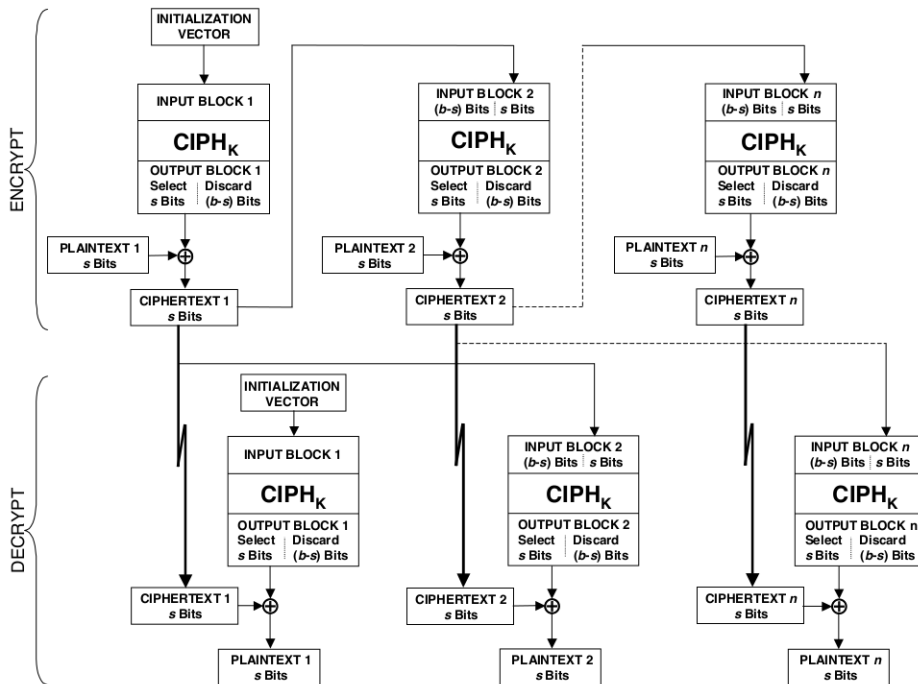
x^y	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
1	1	0	3	2	5	4	7	6	9	8	b	a	d	c	f	e
2	2	3	0	1	6	7	4	5	a	b	8	9	e	f	c	d
3	3	2	1	0	7	6	5	4	b	a	9	8	f	e	d	c
4	4	5	6	7	0	1	2	3	c	d	e	f	8	9	a	b
5	5	4	7	6	1	0	3	2	d	c	f	e	9	8	b	a
6	6	7	4	5	2	3	0	1	e	f	c	d	a	b	8	9
7	7	6	5	4	3	2	1	0	f	e	d	c	b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

4.10. Vecteur d'initialisation

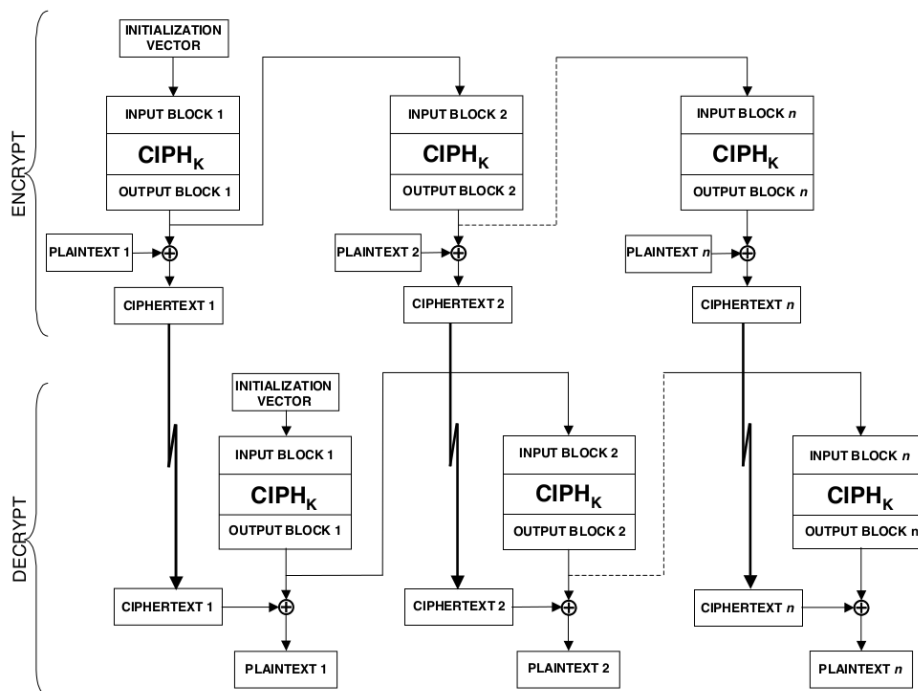
- Pour éviter deux premiers blocs identiques soient chiffrés exactement de la même façon, on ajoute un vecteur d'initialisation
- L'IV n'a pas besoin d'être secret
- L'IV ne doit pas être prédictible
 - Générateur pseudo-aléatoire de qualité cryptographique

- Chiffrer une suite prédictible

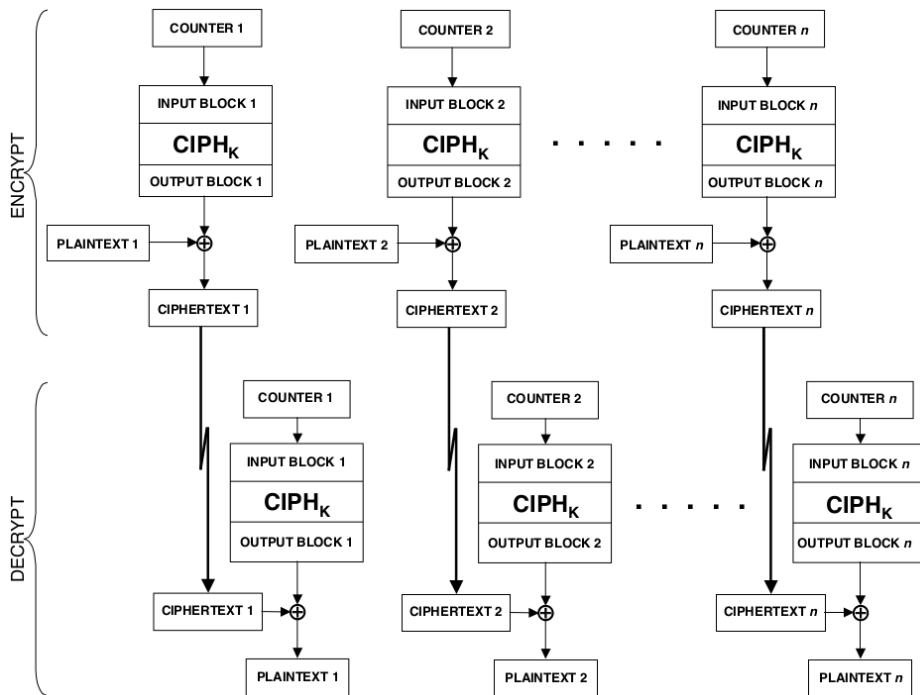
4.11. CFB (Cypher Feed Back)



4.12. OFB (Output Feed Back)



4.13. CTR (Counter)



4.14. Génération de compteurs

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

4.15. 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

```
feed b413 5148 725a a122 976b 272e 1163 b778 c168 526d db65 706b 472d
b269 3e26 f268 8062 5b7c 706e b246 cb44 3e1e 0250 eb49 ce4f 1a52 cf14
7259 5f54 e955 2845 ea57 8f13 5f5d 2b5b bb61 8161 2b7c 4e78 a17f dd2d
126e cd24 7362 5d72 ff67 3b75 ae6d 8b23 7d45 2f2e
```

4.16. Tables de calculs liées à AES dans CTR

input	feed	feee	feef	fef0	fef1	fef2	fef3	fef4	fef5	fef6	fef7	fef8	fef9	fefa	fefb	fefc	fefd	feff	ff00	ff01	ff02	ff03	ff04	ff05	
output	f033	3038	003f	d202	e205	420e	620c	c20a	a20d	720b	b204	1207	2201	9200	5206	8209	f203	3208	020f	db32	eb35	4b3e	6b3c	cb3a	ab3d

ff06	ff07	ff08	ff09	ffa0	ffa1	ffa2	ffa3	ffa4	ffa5	ffa6	ffa7	ffa8	ffa9	ffa0	ffa1	ffa2	ffa3	ffa4	ffa5	ffa6	ffa7	ffa8	ffa9	ffa0	ffa1	ffa2	ffa3	ffa4	ffa5	ffa6	ffa7
7b3b	bb34	1b37	2b31	9b30	5b36	8b39	fb33	3b38	0b3f	de02	ee05	4e0e	6e0c	ce0a	ae0d	7e0b	be04	1e07	2e01	9e00	5e06	8e09	fe03	3e08	0e0f	de02	ee05	4e0e	6e0c	ce0a	ae0d

4.17. 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 : "Salut Mathieu !"
- Essayez de percer le mystère !!

```
Yohan: fac3eb8d0497bcc3158c0c8fc18ffdc809ce5280c889bdc98b9a6192b79a5a853493979f089f
aa96778819d7c195f2d846de67878284b68b928d2792f8ab7ecd6da7
Mathieu: fac3f79901c2a582319e5897c79ffa9c49806cc58188f887d8962880b78b56836792d2dc14
82b59a39895794c28ef39d079f799b8291
Yohan: fac3fd8a0e87ab97319b1d8acd84fcc40883798c9bc4bc9cd88c6794e7db5dd0269f979a069b
b3d32791129ec9dae39d078e72818e80a4d9939a6198eea5
Yohan: fac3fb891d9ae8922d845894888be59d5b8b6a8a8690f890d89b7a8ee28d52822898c3dc129c
a2d324920583c2dae39d078c72968899a79c998c62df
Mael: fac3ed820dc2ba863b821597cd84fb8d08cf
Yohan: fac3f79901c2a9c3348c5896dd8ffb9c418176c5dfc4bc8cd8832884ef9417922898c28f479b
abd3319c02838788e2884890738782d49b9c8196
Yohan: fac3fd98488ba4c321cd19c7d886ed8146ce7c80c896bd9997817b84e4db5491249ed29914dc
```

4.18. Comparaison des modes opératoires

Les critères de comparaison

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

4.19. 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 !!!

4.20. 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)
```

4.21. 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");
        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;
    }
}
```

```
}  
}
```

4.22. 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 : 836a5ec73a8b713711d15e3adb8388cb94fb9ea0e5e509c70a05e0b74603f6ac
Vecteur d'initialisation (IV) utilisé : 68f4aa5e7e8cf1dc7f9a6f468ca0d026

```
dc3678ac3ba8384c0a04245aac6cfae6b5a860d90f7d2ab11849daec34573cb2682b8bcb5240a791790  
03e4227347102  
1b5c3016c3b5a314337211b348d799cb19c1e28ef8d3c196534cad97c09c925fa5199e946f10782f31a  
40b7c3de5506c  
7e16207b68fb37fb4930e991f07f9f0aed593564c36c25fbf9a703ccf54d4e07fbc323e9c66d3b3fdb0  
be3d0e03f1085  
bb375db22671f671299f789bf3dc9200bee14cac41b959528f7272a745f7a4ed8f3ceae300e13c94907  
99c4ef48f16b7  
ae49ca1b67a11c6afd339b53f4859ab432dfc37129fb4e33abc6a4c7d6bf42fd853770a2bc99b519253  
3f08762c4ccad  
a4b4bbc37e8d42cf0be8b7de3862cdfc26e6505b681d2ced1549f5f8315cbf99ce6010fc7f9e0e1e52b  
e3a91abe92d61  
2f53d99d6903887028619eabd6f203e2fc1fe050210a868c12675715bcea3861742578ee1f005f6276f  
0bb26ed7de9f6  
5e1b433a507032374004a2564122bf231367c7c5f1b89e8a9b346b2dde2fdc6a53a74fa8f267c71bc55  
d8c0941f2809a  
cbf70d36c983a877695072d86b0bbd809c2445d0a0e3aeed88a34df7a03e11125ee03b42a84621f00b6  
47ce826b819a3
```

4.23. 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

4.24. 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

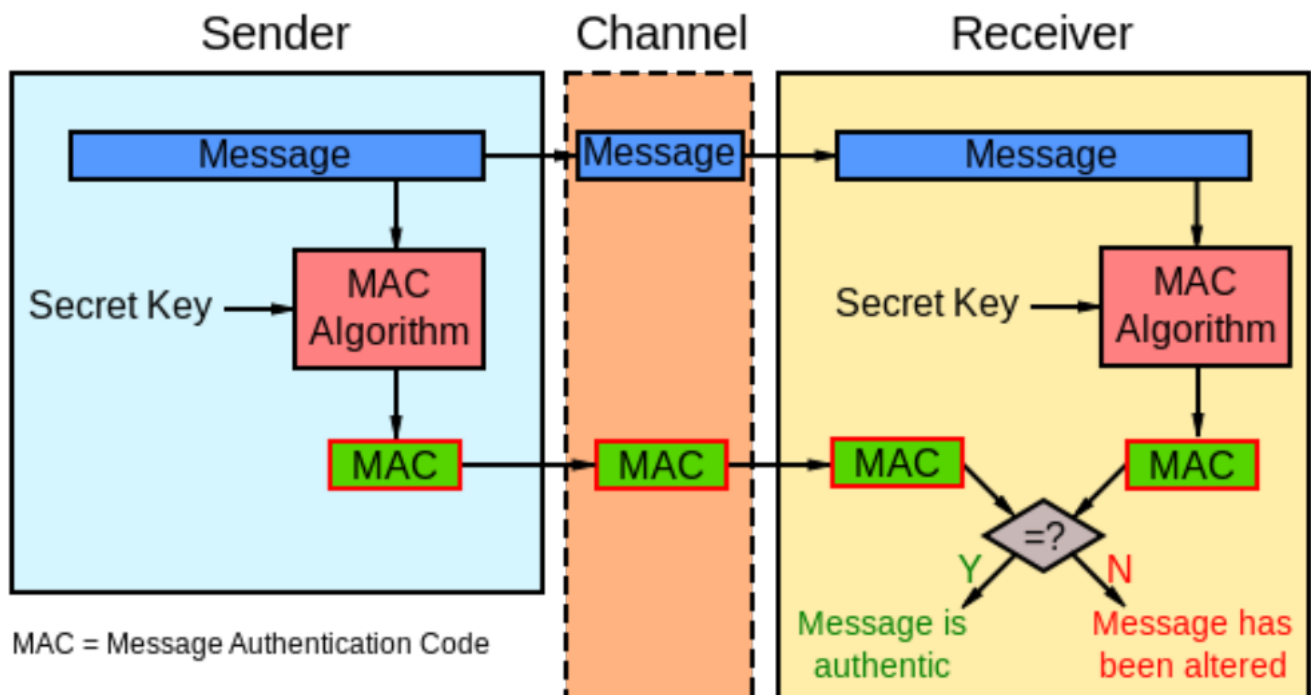
Chapter 5. Intégrité des messages

5.1. 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

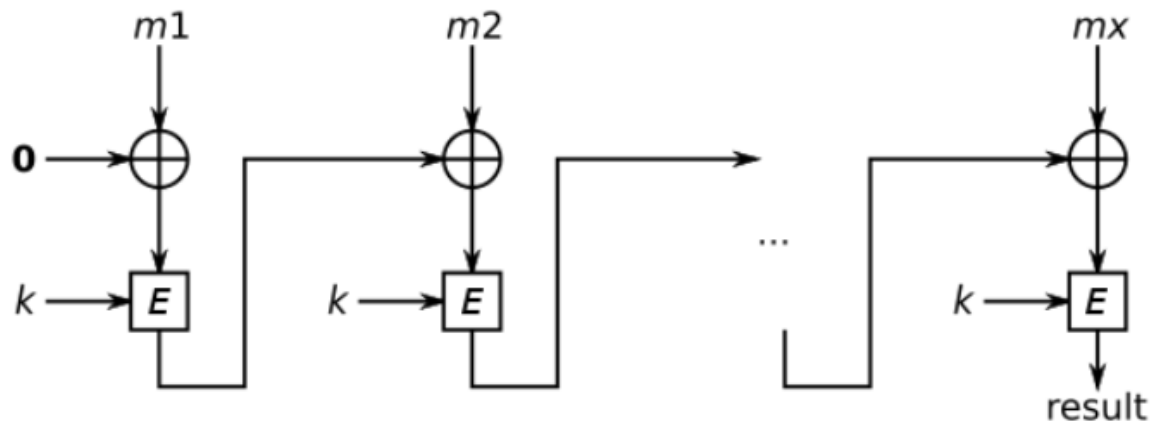
5.2. MAC

Message Authentication Code



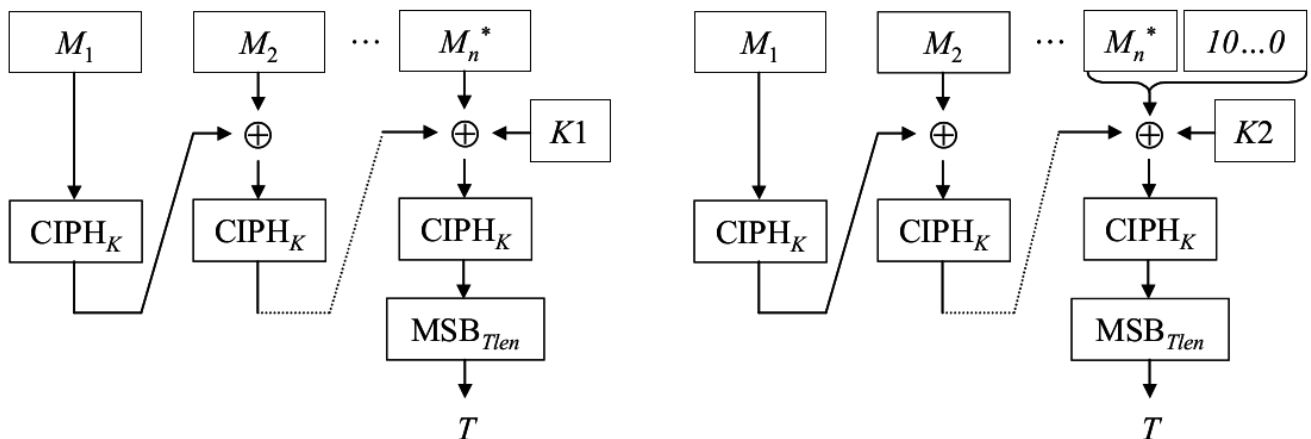
5.3. CBC-MAC

- 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

5.4. Amélioration CMAC - NIST800-38B



5.5. 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 ?

5.6. 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 $H(m)=H(m')$

5.7. 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

5.8. Construction Merkle-Damgard

$$m = m_1 \| m_2 \| \dots \| m_k$$

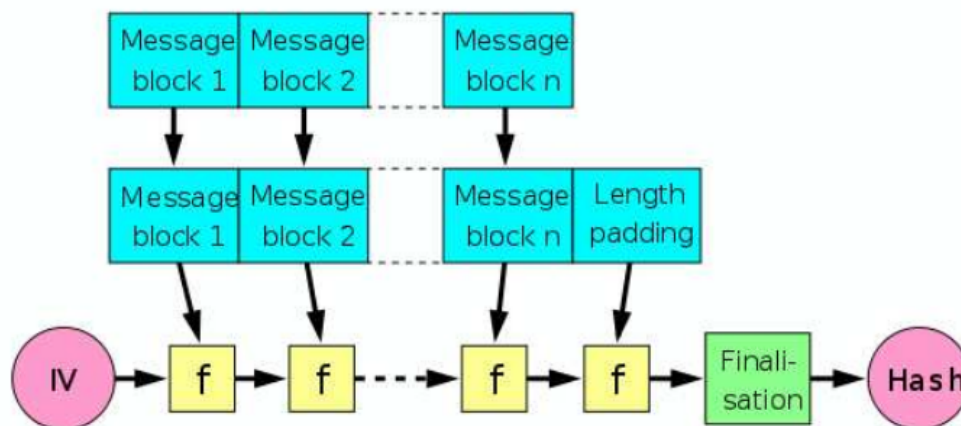
$$H_0 = IV$$

$$H_i = F(H_{i-1}, m_i)$$

$$H(m) = G(H_k)$$

$$F : \{0, 1\}^s \times \{0, 1\}^b \rightarrow \{0, 1\}^s$$

$$G : \{0, 1\}^s \rightarrow \{0, 1\}^t$$



5.9. Pause exercice

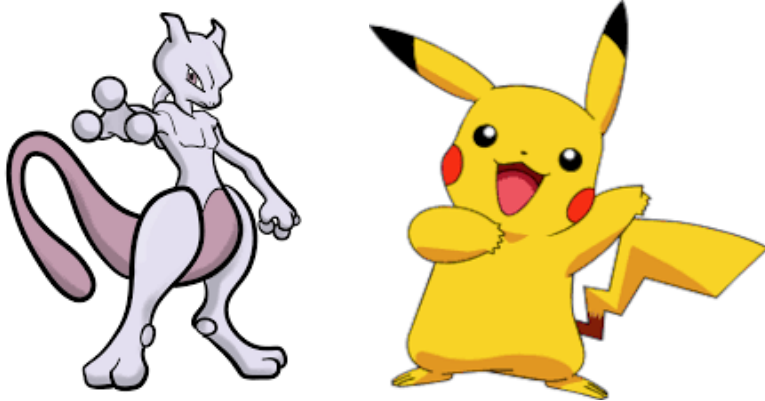
- Charli trouve que la construction MD ressemble beaucoup au chaînage CBC de la semaine dernière.
- Il propose de prendre $IV=0$ et comme fonction de compression le chaînage d'AES-256 avec la clé 0.
- Montrer que cette fonction n'est pas du tout résistante aux collisions !

5.10. 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 $448 \bmod 512$ et enfin on ajoute la longueur initiale du message codée sur 64 bits.

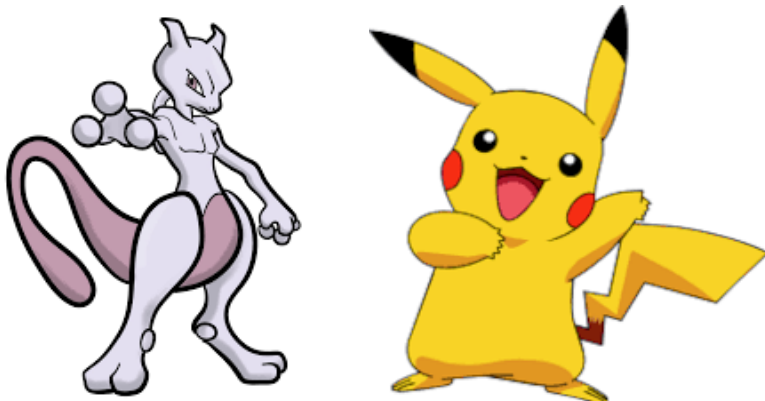
5.11. Application MD5 sur images



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

5.12. Mais...

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



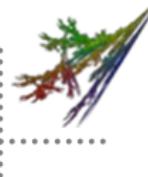
```
md5sum collision*.png
4905e947e3b9542011ab2c1e8721a78f  collision1.png
4905e947e3b9542011ab2c1e8721a78f  collision2.png
```

5.13. MD5 suite des ennuis

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

- 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/>

5.14. MD5 — the end



MD5 Considered Harmful Today

Creating a rogue CA certificate

Alexander Sotirov	<i>New York, USA</i>
Marc Stevens	<i>CWI, Netherlands</i>
Jacob Appelbaum	<i>Noisebridge/Tor, SF</i>
Arjen Lenstra	<i>EPFL, Switzerland</i>
David Molnar	<i>UC Berkeley, USA</i>
Dag Arne Osvik	<i>EPFL, Switzerland</i>
Benne de Weger	<i>TU/e, Netherlands</i>

5.15. 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

5.16. SHA-1 MD5

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

5.17. Que faut il utiliser ?

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

5.18. 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

5.19. HMAC : algorithme

Table 1: The HMAC Algorithm

STEPS	STEP-BY-STEP DESCRIPTION
<i>Step 1</i>	If the length of $K = B$: set $K_0 = K$. Go to step 4.
<i>Step 2</i>	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., $K_0 = H(K) \parallel 00...00$). Go to step 4.
<i>Step 3</i>	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').
<i>Step 4</i>	Exclusive-Or K_0 with <i>ipad</i> to produce a B -byte string: $K_0 \oplus \textit{ipad}$.
<i>Step 5</i>	Append the stream of data ' <i>text</i> ' to the string resulting from step 4: $(K_0 \oplus \textit{ipad}) \parallel \textit{text}$.
<i>Step 6</i>	Apply H to the stream generated in step 5: $H((K_0 \oplus \textit{ipad}) \parallel \textit{text})$.
<i>Step 7</i>	Exclusive-Or K_0 with <i>opad</i> : $K_0 \oplus \textit{opad}$.
<i>Step 8</i>	Append the result from step 6 to step 7: $(K_0 \oplus \textit{opad}) \parallel H((K_0 \oplus \textit{ipad}) \parallel \textit{text})$.
<i>Step 9</i>	Apply H to the result from step 8: $H((K_0 \oplus \textit{opad}) \parallel H((K_0 \oplus \textit{ipad}) \parallel \textit{text}))$.

5.20. 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 6. Canal sûr

6.1. 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

6.2. 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

6.3. 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**

6.4. 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**

6.5. 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

6.6. Conception d'un protocole sécurisé

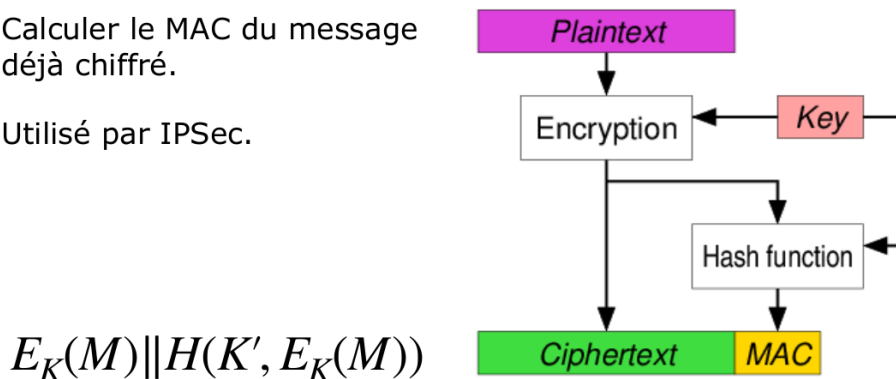
Alice et Bob ont échangé des clés secrètes de 256 bits et élu les algorithmes **AES-CTR-256** et **HMAC-SHA-256**. Ils souhaitent établir un canal sécurisé entre eux. Eve écoute les *échanges sécurisés*.

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

6.7. EtM : Encrypted Then MAC

Calculer le MAC du message déjà chiffré.

Utilisé par IPSec.

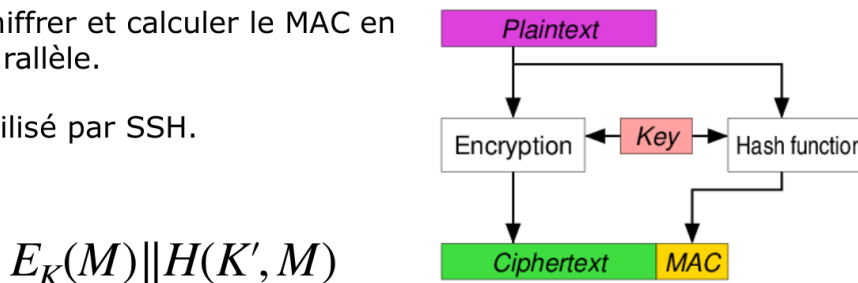


$$E_K(M) || H(K', E_K(M))$$

6.8. E&M : Encrypt and MAC

Chiffrer et calculer le MAC en parallèle.

Utilisé par SSH.

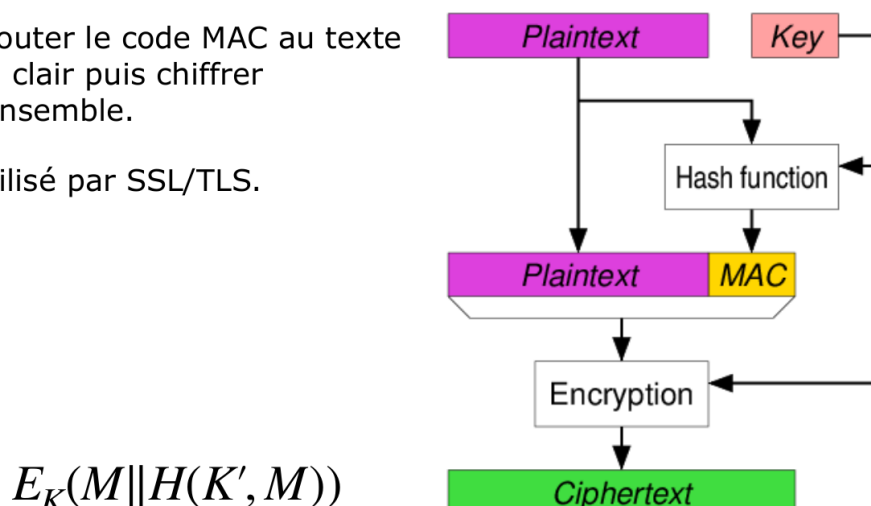


$$E_K(M) || H(K', M)$$

6.9. MtE : MAC then Encrypt

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

Utilisé par SSL/TLS.



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

6.10. 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

6.11. Modes AEAD

- *Authenticated Encryption with Associated Data*
- Proposer des modes opératoires qui apportent le chiffrement et le contrôle d'intégrité tout en assurant la sécurité

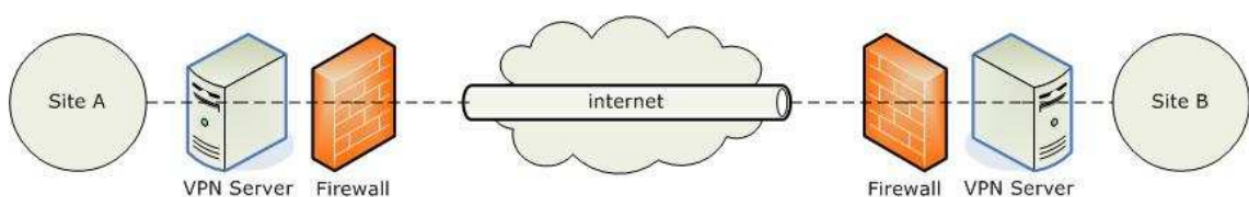
6.12. Mode CCM

- CCM = CTR + CMC-MAC en mode MtE
- Recommandations du NIST : <https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38c.pdf>

$$\begin{aligned}(N, A, P) &\rightarrow B_0, B_1, \dots, B_n \\ Y_0 &= E_K(B_0) \\ Y_i &= E_K(B_i \oplus Y_{i-1}) \quad \forall i \\ T &= Y_n \\ S_0 &= E_K(\text{Ctr}_0) \\ S &= E_K(\text{Ctr}_1) \parallel \dots \parallel E_K(\text{Ctr}_m) \\ C &= (P \oplus T) \parallel (S \oplus S_0)\end{aligned}$$

6.13. En pratique : VPN avec IPsec

- Virtual Private Network permet de créer une liaison sécurisée entre deux réseaux distants à travers un réseau non sûr



6.14. Fonctionnalités

- Authentification : vérifier l'identité des deux extrémités du VPN par authentification mutuelle
- Contrôle d'intégrité : Empêcher la modification du flux réseau qui traverse le VPN
- Confidentialité : Empêcher l'écoute des données
- Technologie intéressante : IPsec en mode transport ou tunnel

6.15. IPsec

- RFC 4301
- Solution normalisée pour le déploiement de VPN IP interopérables

6.16. Composant d'IPsec

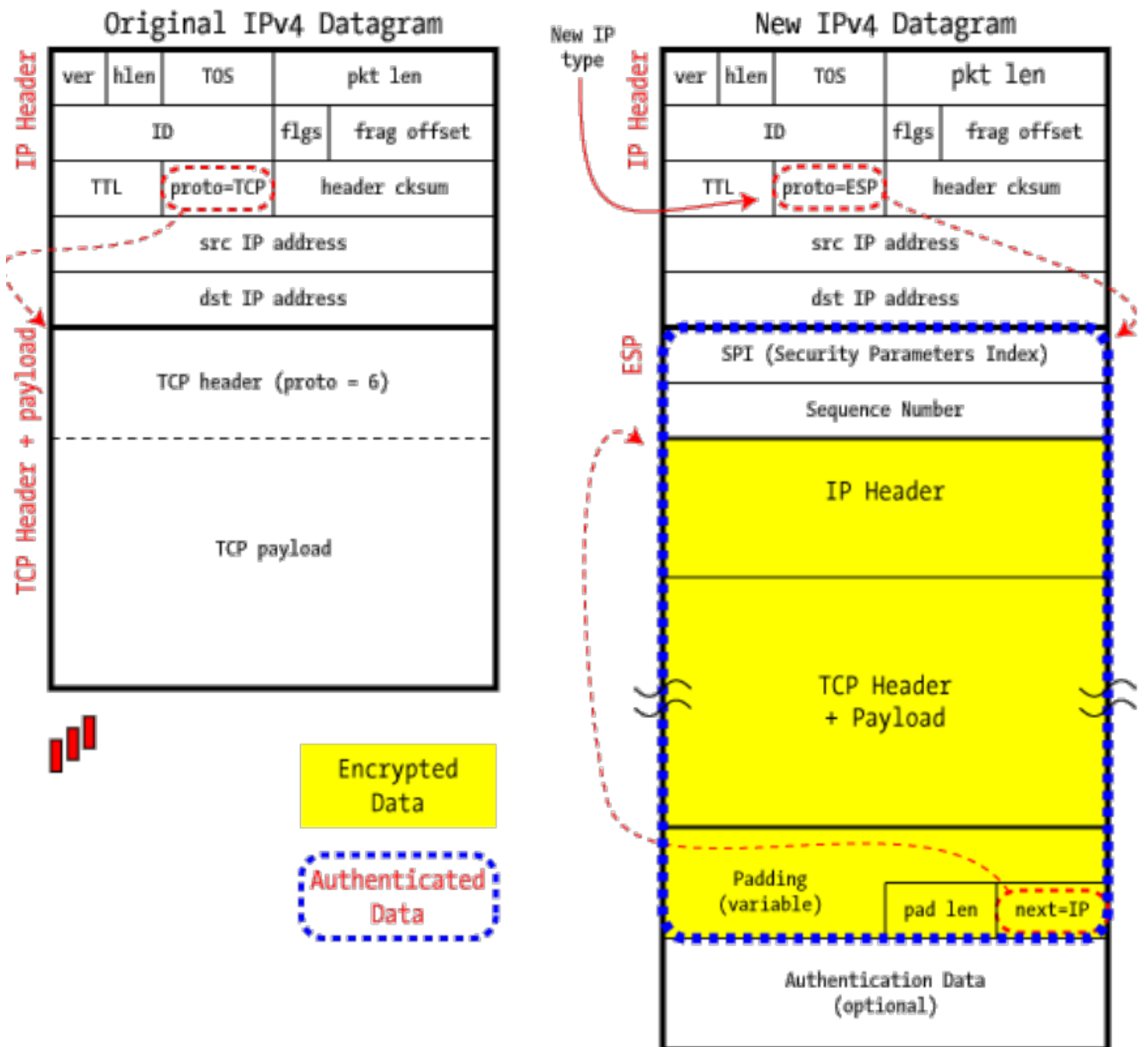
- Protocoles pour le transport de paquets
 - Authentication Header (AH)
 - Encapsulating Security Payload (ESP)
- Une famille d'algorithmes cryptographiques normalisés à combiner à ses protocoles
- Un protocole optionnel de gestion automatique des clés : IKEv2
- Un mécanisme type pare-feu pour décider des paquets qui passent par le tunnel et doivent être encapsulés par AH ou ESP

6.17. AH et ESP

- Encapsulation de chaque paquet IP qui transite par le VPN, nouvelle entête IP + entête AH/ESP
 - Charge utile du paquet IP en mode transport
 - le paquet IP complet en mode tunnel
- Informations présentes dans l'entête
 - SPI : Security Parameters Index
 - SN : Sequence Number
 - ICV : Integrity Check Value
 - Pour en savoir plus : <http://www.unixwiz.net/techtips/iguide-ipsec.html>

6.18. IPsec in ESP Tunnel Mode

IPSec in ESP Tunnel Mode



6.19. Security Policy Database

- Détermine la politique d'encapsulation IPsec du flux réseau
- Règle type pare-feu

```
spdadd 10.0.0.0/24 10.0.1.0/24 any -P in
ipsec esp/tunnel/100.10.10.10-100.20.20.20/require;
spdadd 10.0.1.0/24 10.0.0.0/24 any -P out
ipsec esp/tunnel/100.20.20.20-100.10.10.10/require;
```

6.20. Security Association Database

- Détermine la politique de sécurité à partir du SPI

- Security Association = protocoles + clés + ...

```
add 100.10.10.10 100.20.20.20 esp 1337  
-E des-cbc 0xcafebabec00170ad  
-A hmac-md5 "there is no cake";
```


Chapter 7. Comment partager un secret ?

7.1. 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 ?

7.2. 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

7.3. Protocole de sécurité

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

M, M', \dots des messages

K, K_A, \dots des clés

N, N', \dots des nombres à usage unique

A, B, \dots des agents

$M.M'$ concaténation de M et M'

$\{M\}_K$ chiffrement de M avec la clé K

$A \rightarrow B : M$ A envoie le message M à B

7.4. Protocole de Needham-Schroeder

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

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

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

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

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

7.5. Protocole de Needham-Schroeder

$$A \rightarrow B : A$$

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

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

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

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

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

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

7.6. 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 8. Cryptographie clé publique

8.1. 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).

8.2. 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

8.3. Quelques outils mathématiques : Exponentiation rapide

- Calculez 5^{21} modulo 17 !

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

8.4. 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
		$5 \times 13 \times 1 \equiv 14 \pmod{17}$

8.5. 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

8.6. Quelques outils mathématiques : Euclide étendu

- Calcul de l'inverse de 9 en base 50 ?
- trouver x tel que $x \cdot 9 \bmod 50 = 1$
- Attention $1/9$ n'existe pas dans $\mathbb{Z}/n\mathbb{Z}$

8.7. Détail du calcul

$x = au + bv$	u	v	
$a=50$	1	0	
$b=9$	0	1	soustraire 5x
5	1	-5	soustraire 1x
4	-1	6	soustraire 1x
1	2	-11	
$2a - 11b = 1$ donc $9x-11 = 9 \times 39 = 1 \pmod{50}$			

8.8. Théorème d'Euler

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

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

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

Théorème de Fermat-Euler

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

8.9. 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*

8.10. Des maths... oui encore

- Pour tout entier $n \geq 1$: $\{a \mid \text{pgcd}(n, a) = 1\}$ 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. $m \bmod n = g^k \bmod n$
- k est le logarithme discret de m pour la base g module n

8.11. Protocole DH

- Données du protocole :
 - n premier
 - g non nul
 - n et g peuvent être connus publiquement
 - Alice choisit a et transmet $g^a \bmod n$ à Bob
 - Bob choisit b et transmet $g^b \bmod n$ à Alice
 - Alice et Bob calculent $k = (g^a)^b \bmod n = (g^b)^a \bmod n$

8.12. 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}

8.13. Correction

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

8.14. 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

8.15. 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

8.16. 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 $y=g^s \pmod n$
- Pour chiffrer un message $1 < m < n$, Bob choisit aléatoirement k et transmet la paire : $(c,d)=(g^k \pmod n, m * y^k \pmod n)$
- Alice déchiffre le message en calculant : $(c^s)^{-1}d=(g^{sk})^{-1}my^k=m \pmod n$

8.17. 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
 - $y=g^s \pmod n$
 - $c=g^k \pmod n$
 - $d= m * y^k \pmod n$

8.18. 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
- $y=g^s \pmod n = 2^{153} \pmod{467} = 2*256*156*264 = 224$
- $c=g^k \pmod n = 2^{197} \pmod{467} = 2*16*369*264 = 87$
- $d= m * y^k \pmod n = 331 * 224^{197} \pmod{467} = 331*224*352*64*360 = 331 * 367 = 57$
- Bob envoie (87,57)

8.19. 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 $87^{153} \bmod 467 = 87 \cdot 91 \cdot 342 \cdot 433 = 367$
- Alice calcule l'inverse de 367 en base 467 = 14
- Alice calcule $57 \cdot 14 = 331 = m$

8.20. 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

Chapter 9. RSA

9.1. 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

9.2. 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**)

9.3. 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 **m^e mod n** et envoyer le tout à Alice
4. Alice reçoit **c=m^e mod n**. Elle calcule donc **c^d mod n** et obtient ainsi **m**

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

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

9.5. A vous de jouer !

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

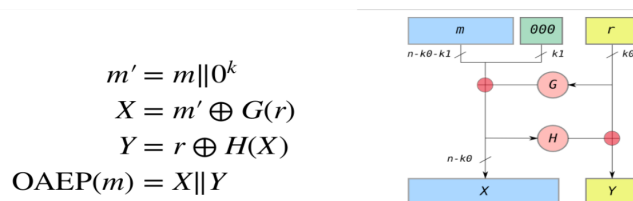
- Déchiffrez le message 8
- On pourra s'aider des carrés successifs modulo 77 : 8, 64, 15, 71, 34, 64,...

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 (N,e) et (N,d) avec $N=pq$
- **Enc** : étant donnés (N,e) et m calcule $c=m^e \pmod N$
- **Dec** : étant donnés (N,d) et c calcule $m=c^d \pmod N$
- Cette façon d'utiliser RSA n'est pas sûre
- Et encore pire en *mode ECB*

9.7. RSA-OAEP

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



9.8. RSA-OAEP par l'exemple

- Soit une taille de blocs ici de 12 bits
- Soit $G = x \rightarrow (x+7)^4 \pmod{2048}$
- Soit $H = x \rightarrow x^2 \pmod{16}$
- Soit m un octet : 138d
- Soit $r=48d$.
- Calculez le message m' construit selon le schéma OAEP qui sera ensuite envoyé au chiffrement RSA

Chapter 10. Signatures

10.1. Schéma de signature numérique

- Repose sur la cryptographie 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

10.2. 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

10.3. Finalement c'est juste l'inverse ?

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

10.4. Formellement

$\text{Gen}(1^k) = (pk, sk)$ algo probabiliste de génération de clés

$\text{Sign}_{sk}(m) = \sigma$ algo probabiliste de signature

$\text{Vrfy}_{pk}(m, \sigma) = b$ algo déterministe de vérification

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

10.5. Hache et signe

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

$$\text{Sign}'_{(sk,s)}(m) = \text{Sign}_{sk}(H^s(m))$$

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

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

10.6. 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 $s = m^d \pmod{N}$
- **Vrfy** : étant donnés (N, e) et m calcule $m = s^e \pmod{N}$

10.7. 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 : $(x*y)^z = x^z * y^z \pmod{N}$

10.8. 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 11. Protocoles : résumé et failles logiques

11.1. 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'
 - **script(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

11.2. Votre expertise ?

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

11.3. 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 ?

11.4. 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 ?

11.5. Votre expertise ?

1. $A \rightarrow S : \text{crypt}(\text{pkS}, A.B)$
 2. $S \rightarrow A : \text{crypt}(\text{pkA}, A.B, \text{pkB.sign}(\text{prvS}, \text{pkB}))$
- Peut-on garantir ici que la clé récupérée **PkB** sera bien la clé de B sachant que S est une entité sûre ?

11.6. Votre expertise ?

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

11.7. 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

11.8. Correction

1. $A \rightarrow Y(B) : g^{na} \bmod n$, Y intercepte le message à destination de B
2. $Y(A) \rightarrow B : g^{ny} \bmod n$, Y envoie un message comme s'il était B
3. $B \rightarrow Y(A) : g^{nb} \bmod n$, Y intercepte le message à destination de A
4. $Y(B) \rightarrow A : g^{ny} \bmod n$, 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.