Sécurité des Systèmes d'Information
Masters Course

Theory: Wednesdays 2pm - 4pm (Battelle A 316-318)
Exercises: Wednesdays 4pm - 6pm (Battelle A 316-318)

Theory: *Eduardo Solana <Eduardo.Solana@unige.ch>*

Exercises: *Joakim Tutt <Joakim.Tutt@unige.ch>*

Alexandre-Quentin Berger
<Alexandre-Quentin.Berger@unige.ch>

Course Objectives

- Learn the basics of information systems security.
- Understand the cryptographic framework allowing to implement security services.
- Describe algorithms, protocols and integrated solutions currently used and understand their weaknesses and how to address them.
- Discover the latest trends on cryptography and information security and the challenges they are facing.
- Learn the terminology enabling access to scientific publications and research reports.
- Motivate students to investigate selected topics on information security and present them to the class in individual presentations.

Tentative Agenda

- Basic Concepts
  - Security Services
  - Internet related Risks
  - Classification of Cryptosystems and Attacks
  - Entropy
  - Random Oracle Model
  - Probabilistic vs. Deterministic Algorithms
  - Computational Security vs. Perfect Secrecy
- Basic Tools
  - (Pseudo-)Random Generation
  - Hash Functions
  - MACs
  - Crypto Algorithms:
    - Stream and Block
    - Symmetric / Asymmetric
  - Certificates
- Access Control and Transaction Security
  - Authentication Protocols
  - Key Establishment Protocols
  - Key and Password Management and Assurance

Tentative Agenda (II)

- Framework
  - Security Policies (BlP, BIBA, Chinese Wall, etc.)
  - MLS
  - OS Security
  - PKI
  - Virtualization and Trusted Computing
- Topology
  - Perimeter/Perimeter-less Security
  - Network Security Protocols
  - Cloud Security
- Topics to be covered (eventually...)
  - Homomorphic Encryption
  - Side Channel Attacks
  - ...

Bibliography

- **[Ros08] Ross J. Anderson. Security Engineering: A Guide to Building Dependable Distributed Systems (2nd Edition). Wiley 2008.**
- **[Sta10] Williams Stallings. Cryptography and Network Security: Principles and Practice (5th Edition). Prentice Hall, 2010.**
- **[Men97]: Menezes, A et al. *Handbook of Applied Cryptography*. CRC series on discrete mathematics and its applications. 1997.**
  URL: `http://cacr.math.uwaterloo.ca/hac/`
- **[Wag03]: Samuel S. Wagstaff, Jr. Cryptanalysis of Number Theoretic Ciphers. Computational Mathematic Series. Chapman & Hall /CRC, 2003.**
- **[Sti06]: Douglas R. Stinson. Cryptography Theory and Practice. (3rd Edition). Chapman & Hall /CRC, 2006.**
- **[Del07] Hans Delfs and Helmut Knebl. Introduction to Cryptography. Principles and Applications (2nd Edition). Springer 2007.**
- **[Kat07] J.Katz and Y. Lindell. Introduction to Modern Cryptography. CRC Press 2007.**

Services de Sécurité

- **Confidentialité**: Protection de l'information d'une divulgation non autorisée
- **Intégrité**: Protection contre la modification non autorisée de l'information
- **Disponibilité**: S'assurer que les ressources sont accessibles aux utilisateurs légitimes
- **Authentification**:
  - **Authentification d'entités**: (*entity authentication*) procédé permettant à une entité d'être sûre de l'identité d'une seconde entité à l'appui d'une évidence corroborant (p.ex.: présence physique, cryptographique, biométrique, etc.). Le terme *identification* est parfois utilisé pour désigner également ce service.
  - **Authentification de l'origine de données**: (*data origin authentication*) procédé permettant à une entité d'être sûre qu'une deuxième entité est la source original d'un ensemble de données. Par définition, ce service assure également l'intégrité de ces données.
- **Non-répudiation**: Offre la garantie qu'une entité ne pourra pas nier être impliquée dans une transaction
- **Non-Duplication**: Protection contre les copies illicites
- **Anonymat** (d'entité ou d'origine de données): Permet de préserver l'identité d'une entité, de la source d'une information ou d'une transaction.

Dangers et Attaques: Synthèse

Services	Dangers	Attaques
Confidentialité	fuite d'informations	écoutes illicites, analyse du trafic
Intégrité	modification de l'information	création, altération ou destruction illicite
Disponibilité	*denial of service*, usage illicite	virus, accès répétés visant à inutiliser un système
Authentification d'entités	accès non autorisés	Vol de mot de passe, faille dans le protocole d'authentification
Authentification de données	falsification d'informations	falsification de signature, faille dans le protocole d'authentification
Non-répudiation	nier la participation à une transaction	prétendre un vol de clé ou une faille dans le protocole de signature
Non-duplication	duplication	falsification, imitation
Anonymat	identification	analyse d'une transaction, accès non autorisés permettant l'identification

Mécanismes de Protection

Services	Mécanismes classiques	Mécanismes digitaux
Confidentialité	scellés, coffre-forts, cadenas	cryptage, autorisation logique
Intégrité	encre spéciale, hologrammes	fonctions à sens unique + cryptage
Disponibilité	contrôle d'accès physique, surveillance vidéo	contrôle d'accès logique, audit, *anti-virus*
Auth. d'entités	présence, voix, pièce d'identité, reconnaissance biométrique	secret + protocole d'authentification, adresse réseau + userid carte à puce + PIN
Auth. de données	sceaux, signature, empreinte digitale	fonctions à sens unique + cryptage
Non-répudiation	sceaux, signature, signature notariale, envoi recommandé	fonctions à sens unique + cryptage + signature digitale
Non-duplication	encre spéciale, hologrammes, tatouage	tatouage digital (*watermarks*), verrouillage cryptographique
Anonymat	brouilleur de voix, déguisement, argent liquide	*mixers*, *remailers*, argent électronique, *deep web*

Risques Liés à Internet

- Programmes malveillants transmis par e-mail (*e-mail malware*)
- Programmes malveillants transmis sur le web (*web malware*)
- Hameçonnage (*Phising*)
- Pourriels (*spam*)
- Rançongiciels (*ransomware*)
- Attaques sur les dispositifs “Internet des Objets” (*Internet of Things/ IoT related attacks*)
- Modification illicite des informations publiées *(information spoofing and website defacement*)
- Attaques dénis de service (*denial of service* ou *DDoS*)
- ...

Liste Non Exhaustive!

Excellente source d’information pour le sujet: **Centrale d’enregistrement et d’analyse pour la sûreté de l’information - MELANI (*https://www.melani.admin.ch*)**

Programmes Malveillants Transmis par ***E-Mail***

- Aussi appelés **maliciels** ou ***malware***
- E-mails conçus pour **inciter le destinataire** à **ouvrir une pièce jointe** ou à **suivre un lien** contenant de la publicité non souhaitée, des informations offensives, des programmes à risque, etc.
- Souvent **ciblés** sur la base des intérêts de la victime (travail préliminaire d'ingénierie sociale (*social engineering*))
- **Conséquences**:
  - Installation de *malware* (*ransomware*, *keyloggers*, etc.) dans le système de la victime *(ordinateur, tablette, smartphone, smartwatch, etc.)*
  - Destruction de données contenues dans l'ordinateur
  - Vol d'informations ou de données personnelles
  - Détournement du système pour des fins malicieuses (p.ex.: minage illicite de *bitcoins*)
  - Diffusion de *malware* (éventuellement à d'autres utilisateurs)

Programmes Malveillants Transmis sur le ***Web***

- Cette technique, souvent appelée ***drive-by download***, permet d'**infecter le système** *(ordinateur, tablette, smartphone, smartwatch, etc.)* **sur lequel s'exécute un client web lors de la simple visite d'un site**
- Il peut s'agir soit
  - d'un site malicieux qui contient le *malware*
  - d'un site web légitime qui aurait été infecté au préalable (par exemple, moyennant une technique appelée *cross-site scripting)*. L'infection pouvant affecter seulement certaines pages...
- La sensibilisation des utilisateurs (ne pas visiter des sites douteux) diminue l'efficacité de cette technique dans la transmission de *malware*
- Les conséquences sont semblables à celle des transmissions par e-mail (voir page 10)
- L'exécution restreinte des scripts (*java/javascript*) dans le navigateur peut limiter la portée de l'infection mais risque de contraindre la navigation dans certains sites...

Hameçonnage (*Phising*)

- Le mot ***phising*** se compose des mots anglais "***password***" (mot de passe), "***harvesting***" (moisson) et "***fishing***" (pêche)
- Cette composition de mots illustre le but principal de cette technique qui consiste **à récolter un maximum d'informations privées** des utilisateurs via des mécanismes de "pêche indiscriminée"
- Lorsque la pêche aux informations est ciblée vers une personne ou organisation spécifique, la technique est dénommée ***spear phising*** (qui provient de *spear fishing* ou pêche au harpon)
- Le vecteur de transmission consiste normalement dans un **e-mail avec une adresse d'expédition falsifiée** (mais souvent indétectable...) qui demande à la victime de fournir des informations privées: adresses e-mail, identifiants (*twitter*, *facebook*, etc.), mots de passe, numéros d'identité, numéros de comptes bancaires, etc.
- Les prétextes utilisés sont variés (mise à jour du système informatique, arrêt du service, retrait d'un envoi, etc.) et vont jusqu'à menacer l'utilisateur en cas de refus

Pourriels (*Spam*)

- Englobe tous les **e-mails indésirables** (souvent publicitaires) reçus par les personnes et les organisations
- Terme utilisé également pour désigner les **pages/*fenêtres pop-up* affichées sans le consentement de l'utilisateur** lors de la navigation web
- On estime que **60%** des e-mails qui circulent dans le monde appartiennent à cette catégorie
- Les conséquences sont souvent limitées à la consommation de ressources de calcul et stockage ainsi qu'au gaspillage de temps associé à la lecture et traitement de ces messages mais...
- ... certains **e-mails spam peuvent également constituer des vecteurs de transmission de malware**
- Ils ont tendance à cibler plus particulièrement les adresses e-mail courtes (p.ex: *abc@gmail.com*) mais fonctionnent également sur la base des listes (souvent échangées / vendues) contenant tous types d'adresses
- Les opérations de filtrage *anti-spam* entraînent des coûts considérables pour les organisations

Rançongiciels (*Ransomware*)

- Cette famille spécifique de malware appartient à la catégorie appelée **Chevaux de Troie** (*Trojan Horses*)
- Leur comportement plus habituel consiste à **chiffrer les données de la victime (**locaux et distants**) afin de les rendre totalement inaccessibles**
- Un message apparait ensuite pour demander le paiement d'une rançon (souvent en *bitcoins* ou une autre monnaie virtuelle) permettant potentiellement de récupérer l'accès aux fichiers chiffrés
- Ils peuvent rester en état ***dormant*** dans le système infecté et être déclenchés par un événement spécifique ou à une date donnée (attaques synchronisées)
- Leurs vecteurs d'infection sont variés mais les **e-mails contenant des pièces jointes malicieuses** sont souvent mis en cause lors des infections primaires
- Des nombreuses variantes existent et continuent à se développer
- On observe parfois d'autres comportements associés à ces *malware*: **dénis de service**, **extorsions ciblées**, **menaces**, etc.

Attaques sur les Dispositifs ***Internet of Things (IoT)***

- Ciblent les **objets connectés** de toute sorte (*caméras*, *TVs*, *frigos*, *capteurs et interrupteurs domotiques*, *installations d'alarme*, etc.)
- Ils sont souvent **plus faciles à pirater que les systèmes traditionnels** par cause de:
  - nombreuses failles de sécurité souvent connues des attaquants
  - mots de passe par défaut
  - négligence de la part des utilisateurs qui ignorent les risques qui leurs sont propres
- La **prise de contrôle à distance** de ces appareils par une entité malveillante implique:
  - Une porte d'entrée au réseau domestique/corporatif
  - Un dispositif pouvant être utilisé pour des activités illicites (*hacking*, attaques *DDoS*, minage de *bitcoins*, etc.)
- L'établissement d'un répertoire précis de tous les dispositifs de ce type connectés au réseau est nécessaire!

Modification Illicite **des Informations Publiées** *Information Spoofing and Website Defacement*

- Attaques visant **l'intégrité** de l'information publiée dans les sites web, les réseaux sociaux, etc.
- Elles portent atteinte **à la réputation et peuvent provoquer d'importants dommages économiques** pour les sociétés ayant une présence Internet
- Dans le cas des **sites web**, la **sécurisation du système hôte** est essentielle ainsi qu'une **configuration aussi restrictive que possible**. Des audits des sécurité récurrents sont vivement recommandés.
- La **protection des informations affichées dans les réseaux sociaux** dépend directement du processus d'authentification permettant d'accéder au profil à risque:
  - Éviter les mots de passe trop simples
  - Privilégier l'authentification forte, si possible *multi-facteur*
  - Fermer proprement les sessions
  - Effacer les *cookies*

Attaques **Dénis de Dervice** (*Denial of Service* / *DDoS*)

- Attaques destinées à **rendre inaccessibles des systèmes informatiques** de toute sorte visant surtout les organisations privées ou étatiques
- Le terme **DDoS** (*Distributed Denial of Service*) désigne une famille d'attaques dans laquelle des multiples (**souvent des dizaines de milliers**) **de dispositifs ciblent simultanément le(s) système(s) victime(s)**
- Le trafique généré atteint plusieurs centaines de gigabits / seconde
- L'efficacité des mécanismes de protection traditionnels (*firewalls*, *sondes* de *prévention* et de *détection d'intrusion*, etc.) est limitée
- L'indisponibilité d'un service peut engendrer:
  - des problèmes **réputationnels**
  - d'importantes **pertes financières** (des **demandes de rançon** peuvent être exigées pour les désactiver)
  - des **hauts risques de sécurité (même physique!)** lorsque des **infrastructures critiques** (hôpitaux, centrales électriques, *backbone* de l'Internet, etc.) sont ciblées

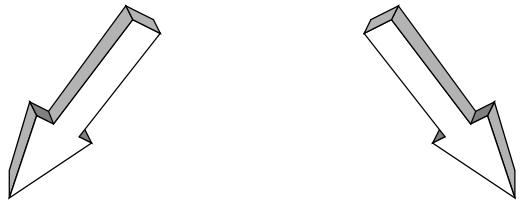
Méthodes Digitales de Sécurité

Problème: Protéger des informations digitales

- dans un environnement distribué
- globalement accessible
- sans frontière matérielle

Solution:

Cryptographie

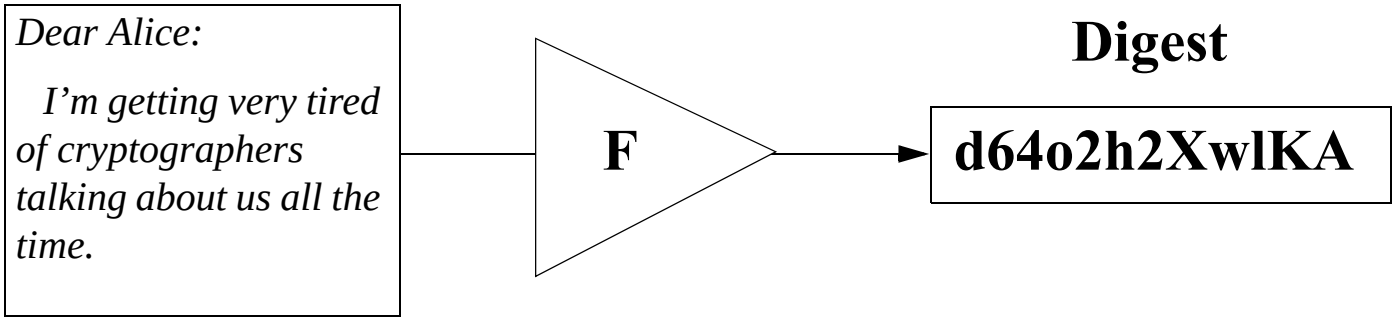


Symétrique **Asymétrique**

+ fonctions à sens unique
+ générateurs (pseudo) aléatoires

Fonctions de Hachage Cryptographiques

Fonctions faciles à calculer dans un sens mais virtuellement impossibles à calculer dans le sens contraire



- Toute modification (*même insignifiante*) du document source se traduit par un *digest* fondamentalement différent.
- Il est virtuellement impossible de retrouver le document source à l'aide seulement du digest (*one-way).*
- Il est virtuellement impossible de retrouver un deuxième document source produisant le même digest (*collision-free).*
- Longueur habituelle des digests: **160 à 512 bits.**
- Les algorithmes à sens unique sont très performants.
- Exemples de fonctions de hachage cryptographiques: **SHA-1, SHA-256, SHA-3,** etc.

Générateurs (Pseudo) Aléatoires

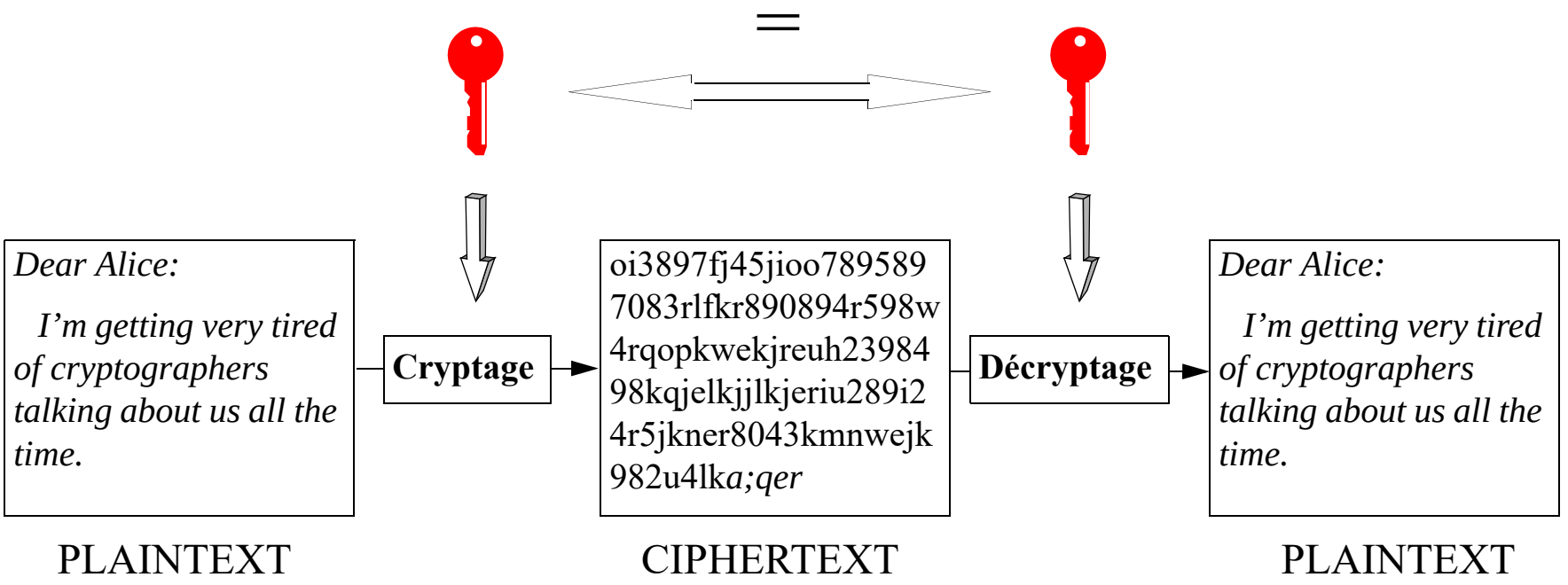
- La génération de nombre aléatoire est un processus très important pouvant compromettre la sécurité d'un bon nombre de systèmes de cryptage.
- Entre autres applications on trouve la génération de clés de session, les vecteurs d'initialisation (DES - CBC mode), des secrets nécessaires à la génération de signatures (ElGamal), etc.
- Un générateur aléatoire (*random generator*) est un dispositif capable de générer des nombres de façon *aléatoire*, *imprévisible* et *non reproductible*. Un tel générateur est normalement doté d'un dispositif externe mesurant des phénomènes physiques connus par leur non-déterminisme (p.ex., une source radioactive ou quantique).
- Les générateurs pseudo-aléatoires sont des procédés déterministes développés à partir d'une séquence aléatoire initial (*seed*) pouvant être obtenue par des méthodes diverses (la fréquence de frappe d'un utilisateur, le nombre d'accès disque, le nombre de paquets reçus par une interface réseau, etc.)
- R. Pitkin dans [Kau95]: "*The use of pseudo-random processes to generate secret quantities can result in pseudo-security*"...
- Description des principaux algorithmes pseudo-aléatoires dans [Sti95].

Cryptographie Symétrique

Aussi appelée cryptographie conventionnelle ou à clés secrètes

(I av. JC, Julius Cesar)

Idée: Sur la base d'une *seule clé secrète*, réaliser une transformation capable respectivement de rendre illisible et de restituer une pièce d'information



Cryptographie Symétrique: Caractéristiques

- Il existe des nombreux systèmes de cryptage symétriques (AES, DES, IDEA, RC4, RC5, etc.) dont certains gratuits et de libre accès.
- Services supportés: *Confidentialité*, *Authentification*, *Intégrité*.
- La clé étant connue des deux intervenants, cette technologie n'offre *pas de support direct pour des signatures digitales*
- La clé étant secrète, elle doit être échangée entre les intervenants par un canal confidentiel alternatif (courrier postal, téléphone, etc.)

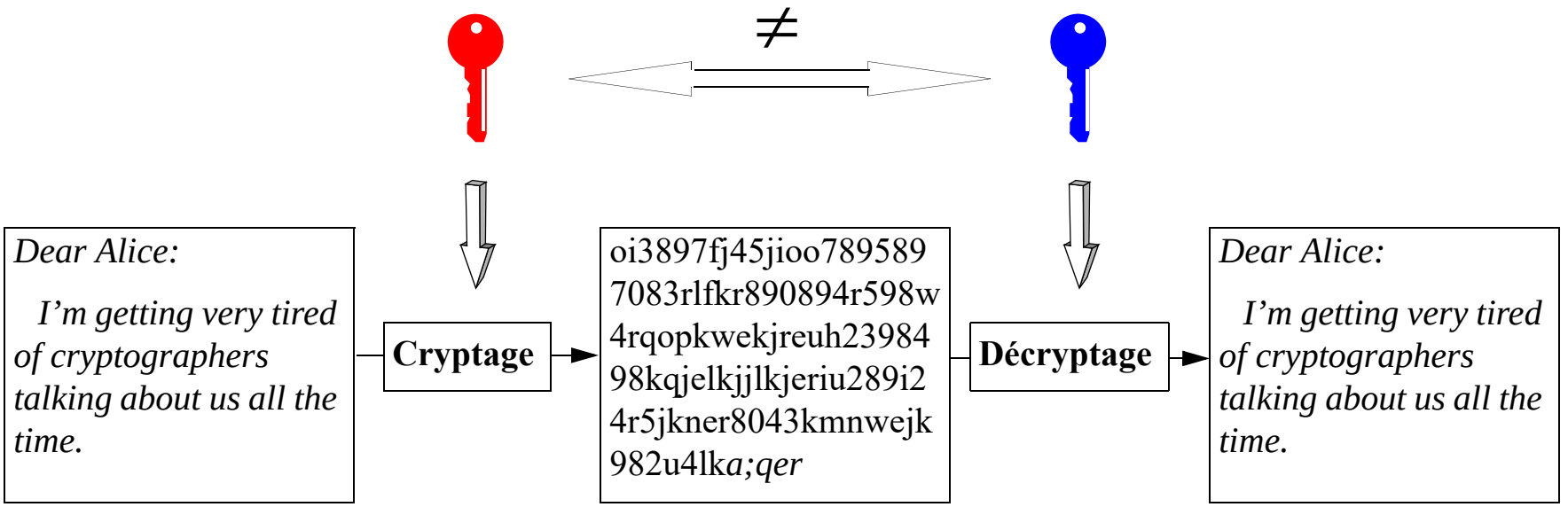
Se prête mieux à la protection de documents personnels qu'à des transactions impliquant des grandes populations (commerce électronique)

Cryptographie Asymétrique

Aussi appelée cryptographie publique ou à clés publiques

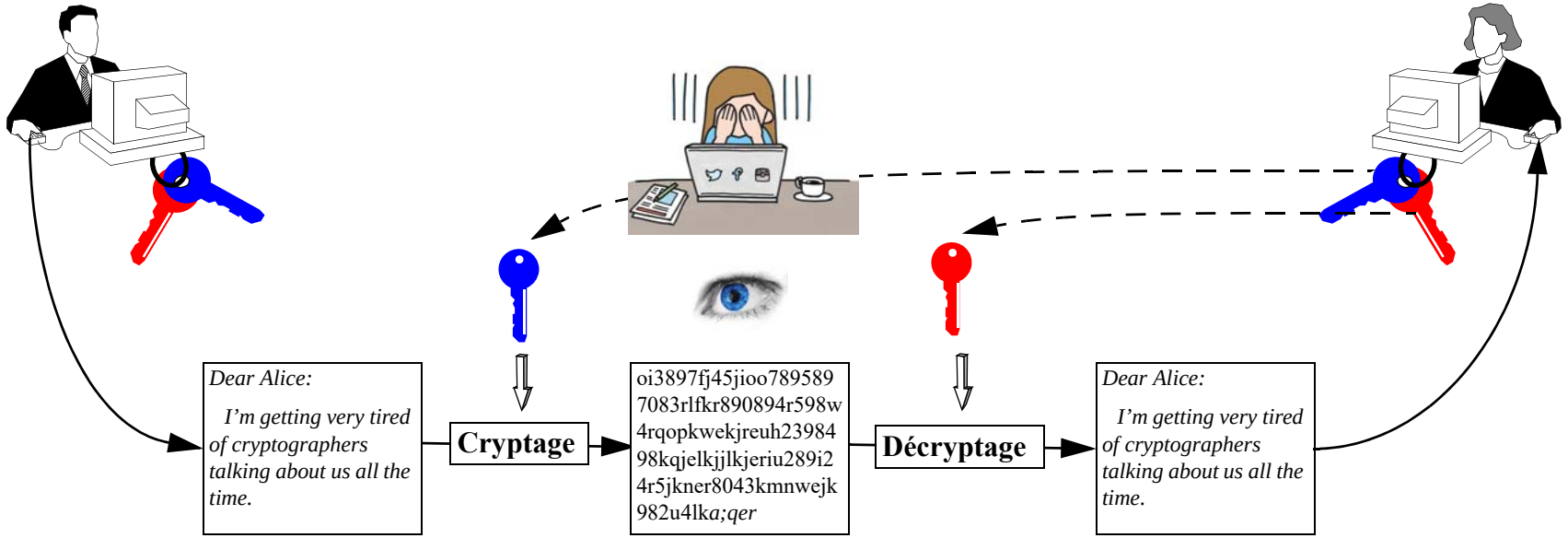
(1976, W. Diffie & M. Hellman)

- **Idée**: Utiliser deux clés différentes - une *secrète* et une *publique* - respectivement pour les opérations de cryptage et décryptage
- Chaque utilisateur dispose d'un *porte-clés* (*keyring*) contenant, au moins, sa clé publique et sa clé privée



Cryptographie Asymétrique: Confidentialité

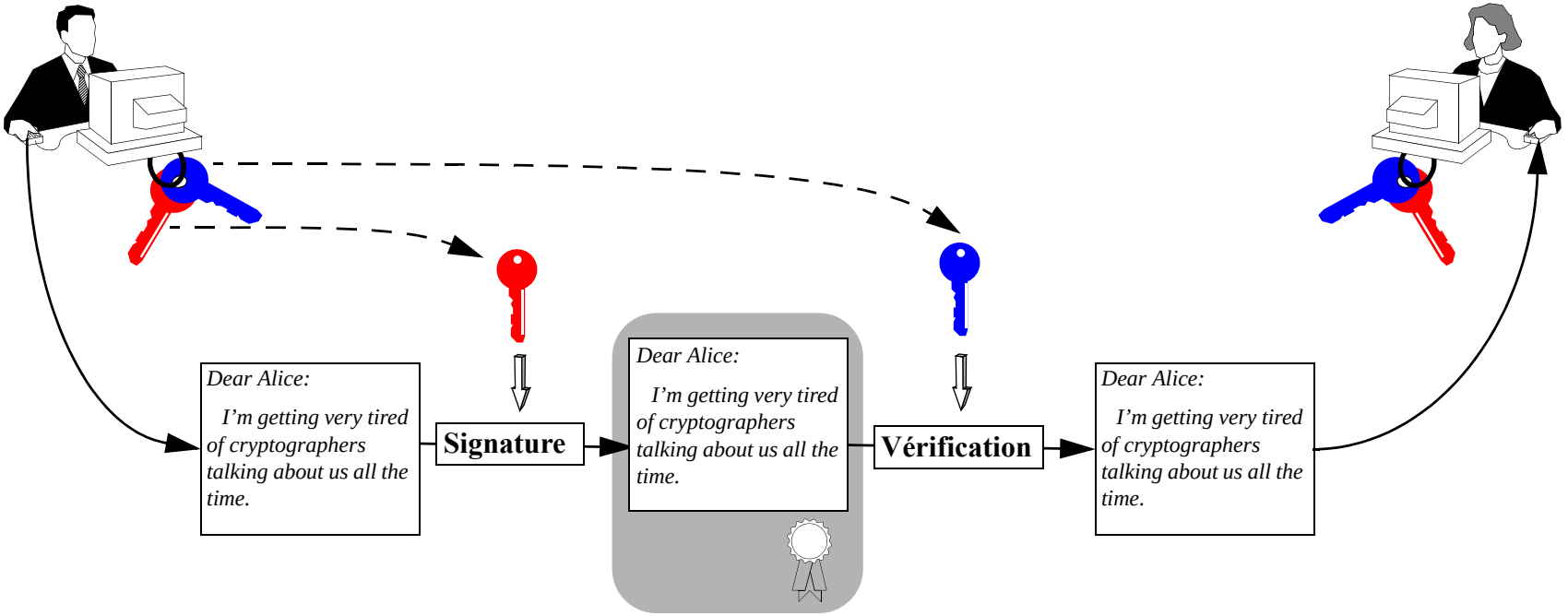
- L'expéditeur crypte l'information avec la clé publique du destinataire (globalement disponible)
- Le destinataire décrypte l'information avec sa clé privée (connue de lui seul)



Seulement les clés (publique et privée) du destinataire sont impliquées dans l'échange confidentiel!

Cryptographie Asymétrique: Signature Digitale

- L'expéditeur signe l'information avec sa clé privée (connue de lui seul)
- Le destinataire vérifie la signature avec la clé publique de l'expéditeur (globalement disponible)

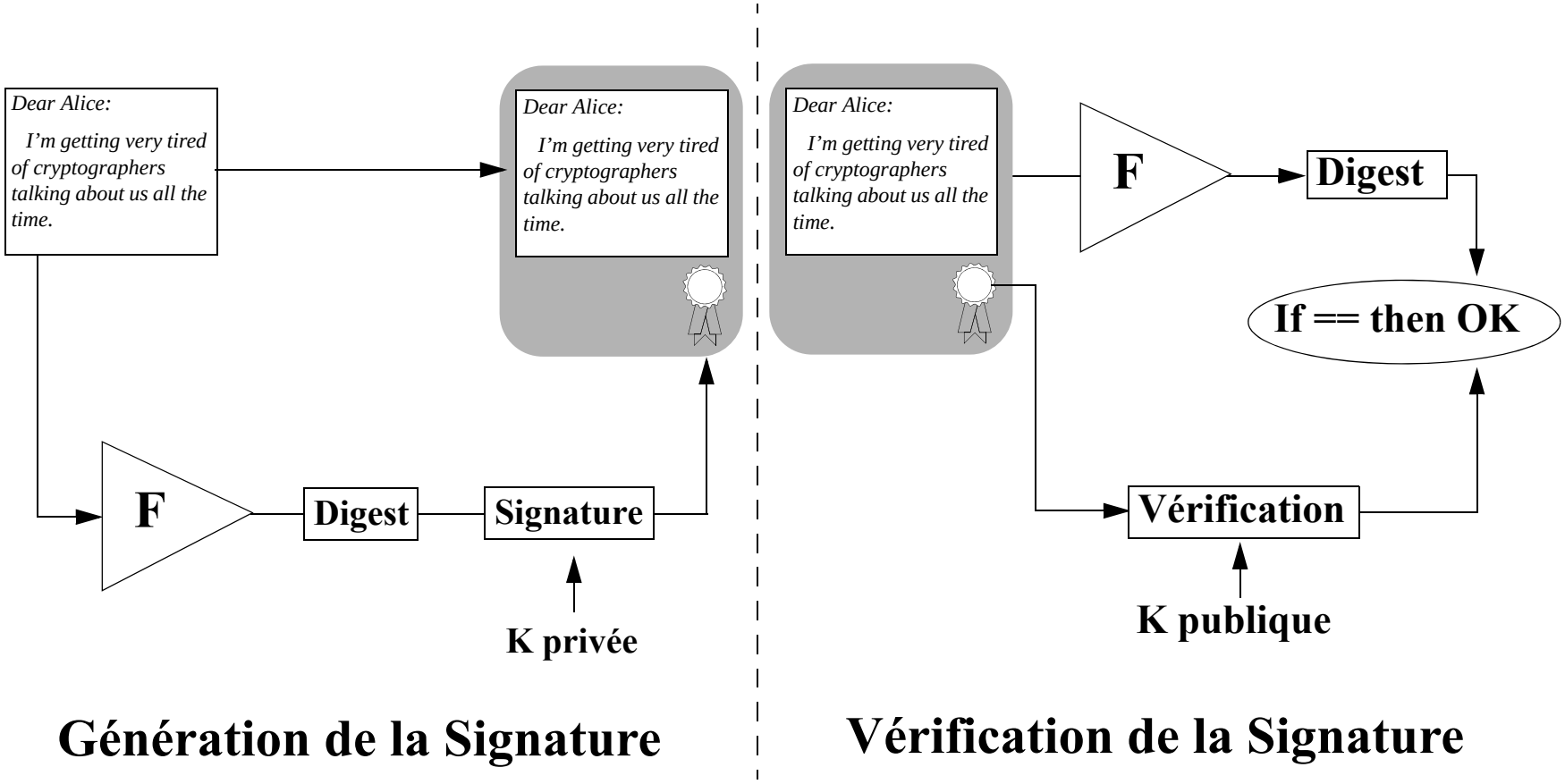


Seulement les clés (publique et privée) de l'expéditeur sont impliquées dans les processus de signature et vérification!

Signature Digitale (II)

Problème: La signature de tout un document est très coûteuse en temps

Solution: Appliquer la signature uniquement au *digest* résultant d'une fonction à sens unique sur tout le document



Exemple d'un Message Signé

-----BEGIN SIGNED MESSAGE----

Dear Alice:

I'm getting very tired of cryptographers talking about us all the time. Why can't they keep their noses in their own affairs?!

Sincerely,

Bob

-----BEGIN SIGNATURE-----

Version: XYZ n.m

iB2FHFbSU7RpAQEqsQMAvo3mETurtUnLBLzCj9/U8oOQg/T7iQcJ

vM8srMV+3VRid64o2h2XwlKAWpfVcC+2v5pba+BPvd86KIP1xRFI

eipmDnMaYP+iVbxxBPVELundZZw7IRE=Xvrc

-----END SIGNATURE-----

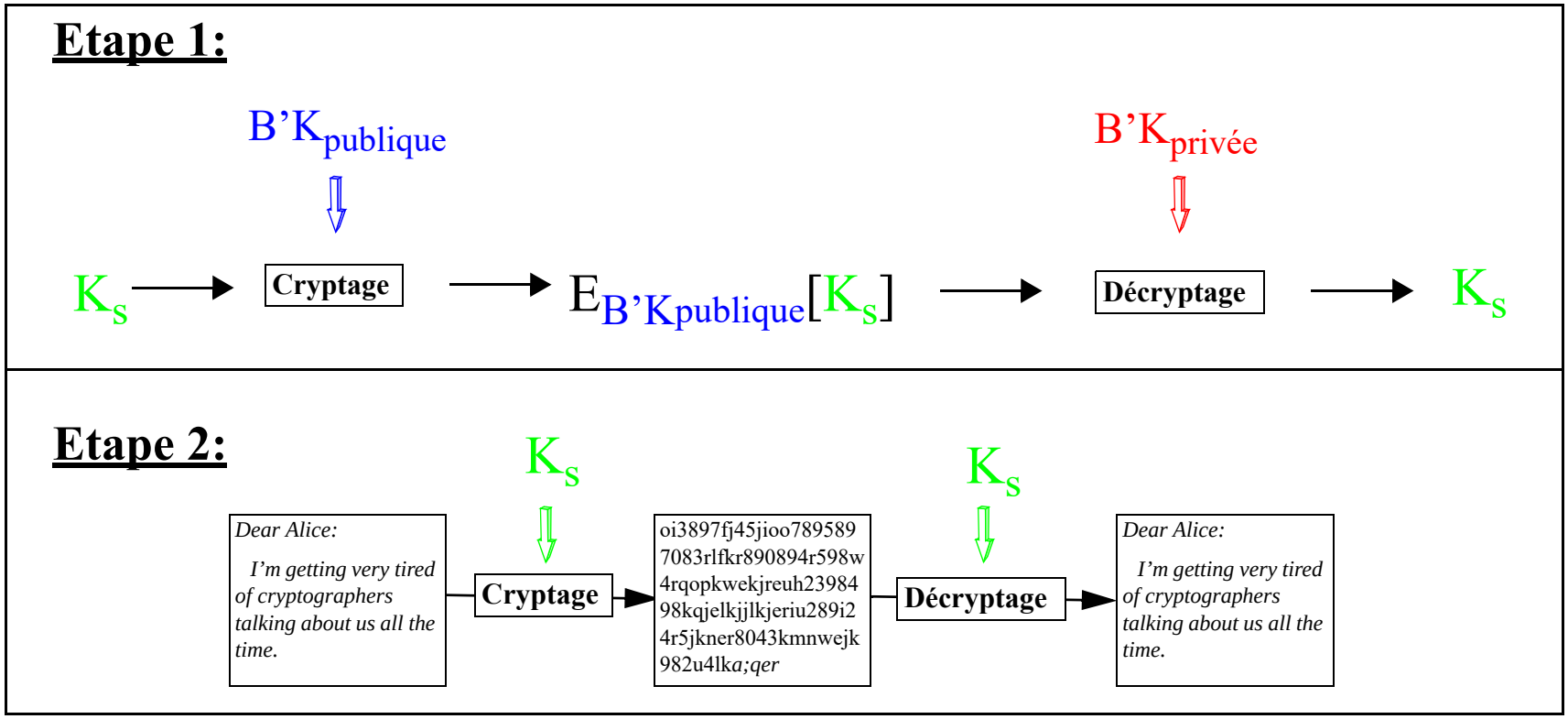
Signatures digitales: Caractéristiques

- La signature change si le document change. La clé privée est toujours la même
- En cas de modification du document ou de la signature, la signature ne sera pas vérifiée (**Intégrité** garantie)
- Il est virtuellement impossible (même pour le détenteur de la clé privée) de générer un deuxième document avec la même signature (la fonction à sens unique est sans collisions)
- Seul le détenteur de la clé privée peut générer une signature qui se vérifie avec la clé publique correspondante

Authentification + Non Répudiation

Crypto asymétrique + Crypto symétrique

Idée: Utiliser la cryptographie publique uniquement pour échanger des clés symétriques



- A génère une clé aléatoire Ks et la transmet à B en l'encryptant avec la clé publique de B (confidentialité)
- A & B communiquent en utilisant la clé Ks pour protéger la confidentialité des échanges

Crypto asymétrique: Fonctionnement (RSA)

Soit n := pq avec p et q deux nombres premiers grands (> 1024 bits)

Soit $\phi(n) = (p-1)(q-1)$

Soit e et d tels que $ed \equiv 1 \bmod \phi(n)$

Par définition des congruences:

$$ed = 1 + k\phi(n)$$

Théorème d'Euler:

$$a^{\phi(n)} \equiv 1 \bmod n$$

Encryption: $C = P^e \bmod n$ **Clé publique: (n,e)**

L'inverse de cette opération est considéré virtuellement impossible à calculer lorsque des grandes nombres son impliqués

Décryption: $P = C^d \bmod n$ **Clé privée: (d)**

Preuve: $(P^{ed} \equiv P^{1 + k\phi(n)} \equiv (P \bmod n)\ (P^{\phi(n)} \bmod n)^k \equiv P \bmod n) \bmod n$

Cryptographie Asymétrique: Conclusions

- Il existe quelques systèmes de cryptage asymétrique (Rabin, ElGamal, etc.) mais le plus utilisé est **RSA**.
- Services supportés: *Confidentialité*, *Authentification*, *Intégrité*, *Signature Digitale & Non-Refus*, *(Non Duplication)*.
- Les opérations liées à la crypto. asymétrique sont jusqu'à 50 fois (!) plus lentes que celles de la crypto. symétrique.
  Une combinaison des deux méthodes est souvent souhaitable
- La distribution des clés est simplifiée par le fait que seules des clés publiques doivent être échangées entre les intervenants (pas besoin d'un canal *confidentiel* alternatif) mais...
- ... il est nécessaire de vérifier que la clé publique appartient réellement au destinataire:
  - Soit le canal d'acquisition de la clé publique est protégé contre toute modification (*authentifié*)
- Soit la clé est ***certifiée*** exacte par un tiers

Cryptographie Symétrique vs. Asymétrique

- Il existe des centaines d'algorithmes symétriques et asymétriques capables de fournir un niveau de confidentialité suffisant.
- Les solutions symétriques offrent les avantages suivants:
  - rapidité (jusqu'à 100 fois plus rapide que les solutions asymétriques)
  - facilité d'implantation en hardware
  - longueur de clé réduite: 128 bits (= 16 caractères => mémorisable !) au lieu de 1024 bits pour des équivalents asymétriques.
- Les solutions asymétriques ont comme arguments principaux:
  - Echange de clés simplifié: les clés doivent être échangées par un canal authentifié mais non-confidentiel.
  - Gestion de clés simplifiée: une seule paire de clés publique/privé suffit à un utilisateur pour recevoir des messages confidentiels de `n` utilisateurs (au lieu de `n` clés différentes dans le cas symétrique).
- Problèmes propres aux deux techniques
  - La gestion de clés par l'utilisateur reste le maillon le plus faible
  - Sécurité (normalement) basée sur des arguments empiriques plutôt que théoriques
  - Restrictions légales d'usage et d'exportation

Cryptographie Symétrique vs. Asymétrique (II)

Activité	Recommandation	Remarques
Protection de documents personnels	Crypto symétrique	Vitesse, clés facilement mémorisables
Protection de documents dans un groupe d'utilisateurs proches	Crypto symétrique	Vitesse, facilité d'échange des clés confidentielles
Etablissement de canaux confidentiels entre utilisateurs distants (inconnus)	Crypto asymétrique	Pas besoin d'avoir un canal confidentiel: authenticité suffit
Transactions entre deux utilisateurs distants, Protection de logiciel (distribution *multicast*)	Crypto asym. pour protection de clé sym. + Crypto sym. pour protection des données	Vitesse, Seule la clé sym. doit être ré-encryptée pour chaque correspondant Copie cryptée du logiciel peut être rendue publique
Protection des segments réseaux	Crypto symétrique	Vitesse, Environnement stable → échange confidentiel des clés facile entre sysadmins.

Dissection d'une Attaque: ***Ransomware***

"*Un **ransomware**, **rançongiciel** ou **logiciel de rançon** est un logiciel malveillant qui prend en otage des données personnelles. Pour ce faire, un **rançongiciel** chiffre des données personnelles puis demande à leur propriétaire d'envoyer de l'argent en échange de la clé qui permettra de les déchiffrer.*" (*Wikipedia* 21 juin 2017).

- Définition incomplète car les ransomware portent sur *un vaste spectre de l'infrastructure informatique*.
- Exemple: l'attaque ransomware par *déni de service distribué (DDoS)* sur *Protonmail* en Novembre 2015[1].
- "***Ransomware Everywhere***" est globalement considérée la menace la plus directe, visible et dangereuse pour utilisateurs et entreprises en 2018!

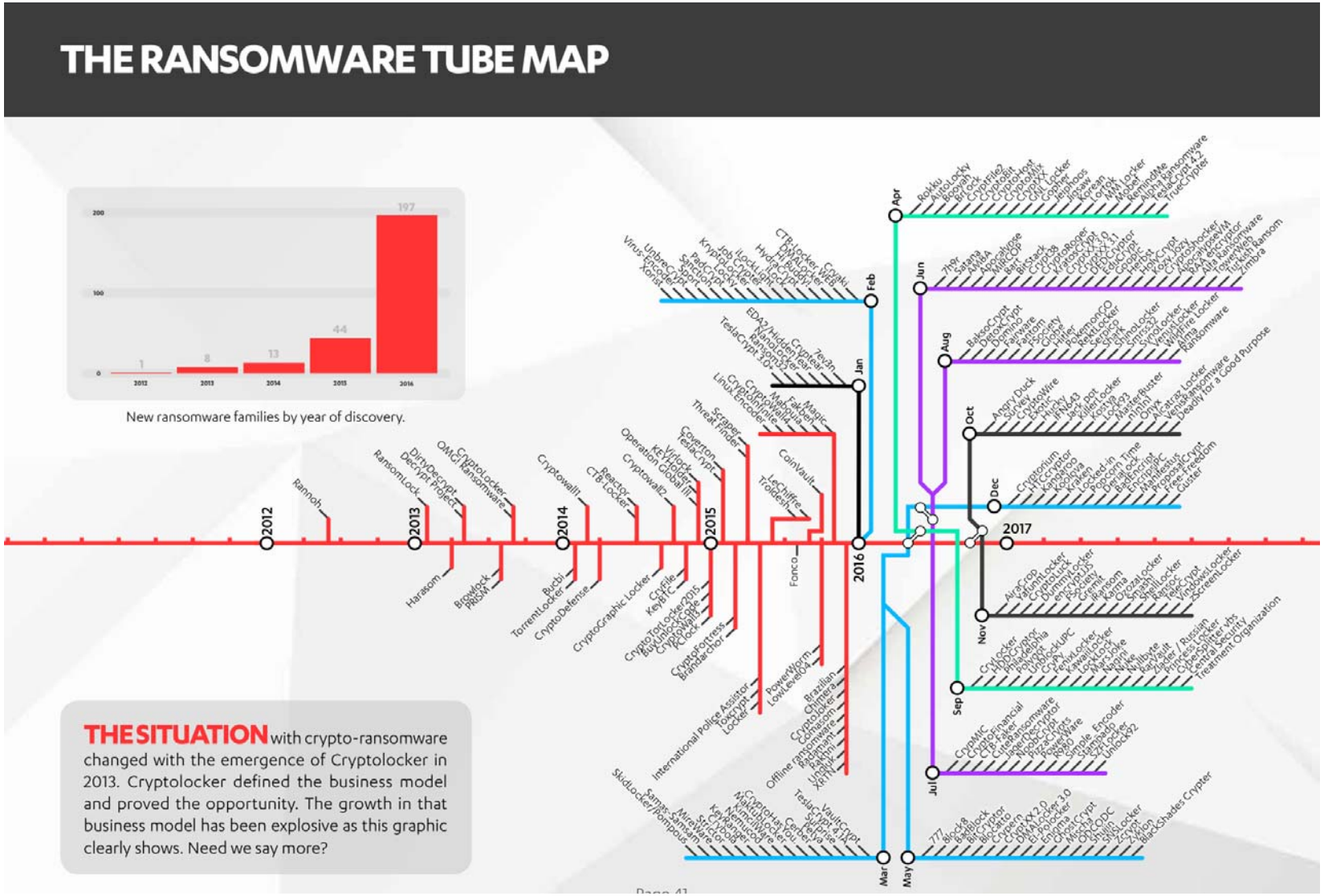
1.https://www.theguardian.com/technology/2015/nov/05/protonmail-service-held-ransom-by-hackers

Ransomware: Quelques Chiffres Globaux[1] [2]

- Les attaques recensées en 2016 ont augmenté de plus de **6'000%** par rapport à 2015 et ont encore **doublé** en 2017.
- Des traces de ransomware étaient trouvées dans **40%** des e-mails *spam* circulant globalement.
- **70%** des entreprises ont payé la rançon exigée par les malfaiteurs. Parmi elles, **50%** ont payé plus de **10'000 US$** et **20%** plus de **40'000 US$**.
- Le chiffre d'affaires du ransomware est estimé à **1 milliard US$** en 2016 mais seul **25%** des attaques sont rendues publiques.
- Le **59%** des infections étaient causées par an e-mail (*Wannacry* se propageait également comme un *ver* via une faille de l'OS).

1. IBM via CNBC: http://www.cnbc.com/2016/12/13/ransomware-spiked-6000-in-2016-and-most-victims-paid-the-hackers-ibm-finds.html
2. Osterman Research Inc.

L'Univers du Ransomware[1]

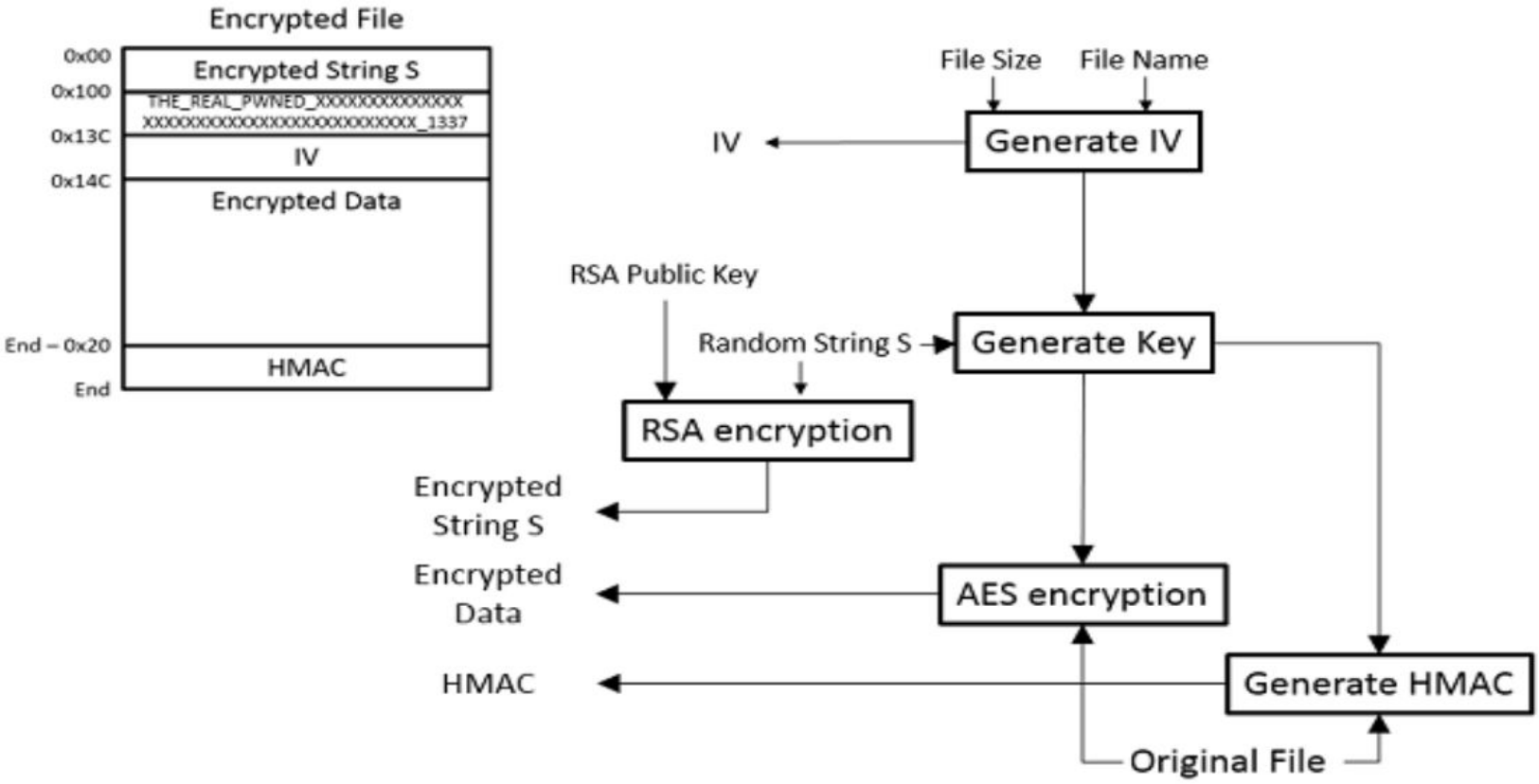


1.2017 State of Cybersecurity. F-Secure Inc.

Ransomware: Vue Intégrale

- Prévision, Remédiation et Réaction
  - *Patching*
  - Détection active et passive (*Firewalls*, *WAFs*, *IDS*, *IPS*, *e-mail malware scan*, etc.)
  - *Backups offline* !
  - Politique de Sécurité - Règles de bon usage de la messagerie
  - Formation !
  - Payer ou pas payer...
- Dissection Technique de l'Attaque
  - Infection et propagation
  - Exécution
  - Paiement (*Crypto-currencies* / *Bitcoin*)
  - Occultation (*Obfuscation*, *TOR Networks/Deep Web*)

Schéma Générique d'un ***Ransomware Cryptolocker***



- Les **clés privées** de déchiffrement sont **stockées dans les serveurs de l'attaquant**
- Elles sont envoyées à la victime après **paiement en *bitcoins***
- La **trace** est **broyée** à l'aide des **réseaux *TOR***

Ransomware Cryptolocker: Cibles[1]

.jin, .xls, .xlsx, .pdf, .doc, .docx, .ppt, .pptx, .txt, .dwg, .bak, .bkf, .pst, .dbx, .zip, .rar, .mdb, .asp, .aspx, .html, .htm, .dbf, .3dm, .3ds, .3fr, .jar, .3g2, .xml, .png, .tif, .3gp, .java, .jpe, .jpeg, .jpg, .jsp, .php, .3pr, .7z, .ab4, .accdb, .accde, .accdr, .accdt, .ach, .kbx, .acr, .act, .adb, .ads, .agdl, .ai, .ait, .al, .apj, .arw, .asf, .asm, .asx, .avi, .awg, .back, .backup, .backupdb, .pbl, .bank, .bay, .bdb, .bgt, .bik, .bkp, .blend, .bpw, .c, .cdf, .cdr, .cdr3, .cdr4, .cdr5, .cdr6, .cdrw, .cdx, .ce1, .ce2, .cer, .cfp, .cgm, .cib, .class, .cls, .cmt, .cpi, .cpp, .cr2, .craw, .crt, .crw, .phtml, .php5, .cs, .csh, .csl, .tib, .csv, .dac, .db, .db3, .dbjournal, .dc2, .dcr, .dcs, .ddd, .ddoc, .ddrw, .dds, .der, .des, .design, .dgc, .djvu, .dng, .dot, .docm, .dotm, .dotx, .drf, .drw, .dtd, .dxb, .dxf, .dxg, .eml, .eps, .erbsql, .erf, .exf, .fdb, .ffd, .fff, .fh, .fmb, .fhd, .fla, .flac, .flv, .fpx, .fxg, .gray, .grey, .gry, .h, .hbk, .hpp, .ibank, .ibd, .ibz, .idx, .iif, .iiq, .incpas, .indd, .kc2, .kdbx, .kdc, .key, .kpdx, .lua, .m, .m4v, .max, .mdc, .mdf, .mef, .mfw, .mmw, .moneywell, .mos, .mov, .mp3, .mp4, .mpg, .mrw, .msg, .myd, .nd, .ndd, .nef, .nk2, .nop, .nrw, .ns2, .ns3, .ns4, .nsd, .nsf, .nsg, .nsh, .nwb, .nx2, .nxl, .nyf, .oab, .obj, .odb, .odc, .odf, .odg, .odm, .odp, .ods, .odt, .oil, .orf, .ost, .otg, .oth, .otp, .ots, .ott, .p12, .p7b, .p7c, .pab, .pages, .pas, .pat, .pcd, .pct, .pdb, .pdd, .pef, .pem, .pfx, .pl, .plc, .pot, .potm, .potx, .ppam, .pps, .ppsm, .ppsx, .pptm, .prf, .ps, .psafe3, .psd, .pspimage, .ptx, .py, .qba, .qbb, .qbm, .qbr, .qbw, .qbx, .qby, .r3d, .raf, .rat, .raw, .rdb, .rm, .rtf, .rw2, .rwl, .rwz, .s3db, .sas7bdat, .say, .sd0, .sda, .sdf, .sldm, .sldx, .sql, .sqlite, .sqlite3, .sqlitedb, .sr2, .srf, .srt, .srw, .st4, .st5, .st6, .st7, .st8, .std, .sti, .stw, .stx, .svg, .swf, .sxc, .sxd, .sxg, .sxi, .sxi, .sxm, .sxw, .tex, .tga, .thm, .tlg, .vob, .war, .wallet, .wav, .wb2, .wmv, .wpd, .wps, .x11, .x3f, .xis, .xla, .xlam, .xlk, .xlm, .xlr, .xlsb, .xlsm, .xlt, .xltm, .xltx, .xlw, .ycbcra, .yuv

---

1.Intel Security. Advanced Threat Research. http://www.intelsecurity.com

Cryptography Basic Notions

- Kerckhoffs' principle
- Cryptosystem classification based on complexity
- Entropy
- Classification of cryptosystems attacks
- Random oracles and encryption oracles
- Deterministic and probabilistic encryption
- Historic cryptosystems
- The *One-Time* Pad
- Steganography

Kerckhoffs' principle

Auguste Kerckhoffs wrote in 1883[1] two journal articles formulating six design principles for military ciphers:

1. **The system must be practically, if not mathematically, indecipherable.**
2. **It should not require secrecy, and it should not be a problem if it falls into enemy hands.**
3. It must be possible to communicate and remember the key without using written notes, and correspondents must be able to change or modify it at will.
4. It must be applicable to telegraph communications.
5. It must be portable, and should not require several persons to handle or operate.
6. Lastly, given the circumstances in which it is to be used, the system must be easy to use and should not be stressful to use or require its users to know and comply with a long list of rules.

Kerckhoffs' states already in the XIX century that the system should be **mathematically proven** and that there is no **security through obscurity !**

1. Auguste Kerckhoffs, *"La cryptographie militaire" Journal des sciences militaires*, vol. IX, pp. 5–83, January 1883, pp. 161–191, February 1883.

Classification des systèmes de cryptage

- **Sécurité inconditionnelle (***unconditional security* aussi appelée *perfect secrecy*):
  - La sécurité du système de cryptage n'est pas compromise par la puissance de calcul destinée à la cryptanalyse.
  - Cette catégorie s'appuie sur la théorie de l'information publiée par Shannon en 1949.
  - Plus précisément, un système de cryptage est *inconditionnellement sûr* si la probabilité de rencontrer un *plaintext* `x` après l'observation du *ciphertext* correspondant `y` est identique à la probabilité *à priori* de rencontrer le plaintext `x`. En d'autres termes, le fait de disposer de couples plaintext/ciphertext `(x,y)` ne constitue aucune aide pour la cryptanalyse.
  - Une condition nécessaire pour qu'un système soit inconditionnellement sûr est que la clé soit au moins de la même taille que le message et, surtout, qu'elle ne soit pas réutilisée pour encrypter des messages différents.
  - Cette condition rend ces systèmes peu adaptés aux besoins cryptographiques habituels et réduit leur domaine d'intérêt à un cadre théorique.
  - L'exemple classique est le *one-time pad* inventé en 1917 par J.Mauborgne and G. Vernam.
  - Fondements théoriques des systèmes inconditionnellement sûrs + d'autres exemples dans [Sti06].

Classification des systèmes de cryptage (II)

- ***As hard as* / équivalent / *provable security***
  - Lorsqu'on peut prouver que la cryptanalyse de l'algorithme est aussi difficile que de résoudre un problème mathématique réputé difficile.
  - Par exemple *la factorisation de grands nombres, le calcul de racines carrées modulo un "composite", le calcul de logarithmes discrets dans un groupe fini*, etc. (voir chapitre sur la Cryptographie Asymétrique).
  - L'algorithme de Rabin et RSA (cas générique[1]) sont "prouvés" équivalents à la factorisation. Une telle preuve s'appelle de "réduction" (*reduction proof*).
  - La notion de *provable security* est à l'origine d'une importante controverse dans le monde cryptographique[2].
- **Sécurité calculatoire** (*computational security* aussi appelé *practical security*)
  - Un système de cryptage est dans cette catégorie si l'effort calculatoire nécessaire à le "casser" en utilisant les meilleures techniques possibles est au delà (avec une marge raisonnable) des ressources de calcul d'un adversaire hypothétique.
  - La grande majorité de systèmes de cryptage symétriques (AES,DES, IDEA, RC4, etc.) sont dans cette catégorie.

---

1. *Breaking RSA Generically is Equivalent to Factoring*. D. Aggarwal and U.Maurer. Eurocrypt 2009.
2. *Another Look at Provable Security.* Neil Koblitz and Alfred Menezes. Cryptology ePrint Archive 2004.

Entropie

- Une définition essentielle dans la cryptographie est la quantité d'information "effective" contenue dans un message.
- Par exemple, les jours de la semaine ("lundi",..., "dimanche") peuvent intuitivement être encodés comme des chaînes de caractères de longueur ≤ len("mercredi"), soit 8 x 8 = 64 bits. Cependant, la quantité d'information effective de la variable "jour de la semaine" peut être encodée de manière optimale sur 3 bits (car $2^3 = 8$ est suffisant pour représenter les 7 variations possibles).
- L'**entropie** (Shannon, 1948) est la formalisation mathématique de cette définition:
- Soit X une variable aléatoire avec un ensemble fini de valeurs possibles $x_1, x_2,..., x_n$ avec probabilité $P(X=x_i) = p_i$, avec $0 \leq p_i \leq 1 \ \forall$ i, t.q. $1 \leq i \leq n$ et t.q. $\sum_{i=0}^{n} p_i = 1$. L'**entropie** de X, H(X) est définie comme: $H(x) = -\sum_{i=1}^{n} p_i \log p_i = \sum_{i=1}^{n} p_i \log(1/p_i)$ avec (par convention) $p_i \cdot \log p_i = p_i \cdot \log(1/pi) = 0$ si $p_i = 0$. Tous les logs à base 2.
- Cette formule permet d'approximer le nombre de bits qui sont nécessaires pour encoder les éléments dans X. La **redondance** est la différence entre le codage effectif d'une pièce d'information et son entropie. Le langage naturel (anglais) a une entropie de 1.3 par lettre et, donc, une redondance de 6.7 bits avec un codage à 8 bits !

Entropie (II)

Propriétés:

(i) $0 \leq H(X) \leq \log n$

(ii) $H(X) = 0$ ssi $\exists$ i t.q. $p_i = 1$ et $p_j = 0$, $\forall$ $j \neq i$ (c.à.d. il n'y a pas d'incertitude sur le résultat)

(iii) $H(X) = \log n$ ssi $p_i = 1/n$ $\forall$ i $1 \leq i \leq n$ (c.à.d. tous les résultats sont équiprobables).

Exemple:

L'entropie de la variable "jours de la semaine" en admettant que toutes les valeurs sont équiprobables serait: $H(\text{jours de la semaine}) = \sum_{i=1}^{7} \frac{1}{7} \log 7 = \log 7 = 2{,}807$

Entropie conditionnelle de X étant donné Y = y:

$$H(X|Y=y) = -\sum_{x} P(X=x|Y=y) \cdot \log(P(X=x|Y=y))$$

Entropie conditionnelle de X étant donné Y:

$$H(X|Y) = \sum_{y} P(Y=y) \cdot H(X|Y=y)$$

L'entropie conditionnelle mesure le degré d'incertitude qui reste sur X (le *plaintext*) après avoir observé Y (le *ciphertext*).

Attaques sur les systèmes de cryptage

- Attaque *ciphertext-only*: L'adversaire (ou le cryptanalyste) essaye de trouver la clé ou le plaintext à partir de l'observation du ciphertext seul. Un système de cryptage vulnérable à cette attaque n'offre aucune sécurité.
- Attaque *known-plaintext*: L'adversaire a des couples plaintext/ciphertext à disposition. Cette attaque est à peine plus facile à mettre en place que le *ciphertext- only.*
- Attaque *chosen-plaintext*: L'adversaire peut choisir le plaintext et obtenir le ciphertext correspondant, le but étant de retrouver du plaintext à partir des ciphertext observés.
- Attaque *adaptive chosen-plaintext*: Il s'agit d'une attaque *chosen-plaintext* ou le choix sur les plaintexts peut dépendre des ciphertexts reçus lors des requêtes précédentes.
- Attaque *chosen-ciphertext*: L'adversaire choisit le ciphertext et obtient le plaintext correspondant. L'objectif de cette attaque étant normalement de trouver la clé.
- Attaque *adaptive chosen-ciphertext*: Il s'agit d'une attaque *chosen-ciphertext* ou le choix sur les ciphertexts peut dépendre des plaintexts reçus lors des requêtes précédentes.

Random Oracles

- A random oracle is an abstract entity (available to legitimate parties and adversaries) that responds to queries containing a given input `x` with perfectly random responses `Orc(x)`.
- The only exception to this behaviour resides in the previously processed inputs $(x_1, x_2, x_3, ..., x_n)$ where the random oracle will provide the same response than the previous query so that `if x1' = x1 then Orc(x1') = Orc(x1)` which means that the random oracle is *deterministic* with previously processed inputs.
- A random oracle can be modeled as a mathematical function `Orc: X` $\rightarrow$ `Y` where $\forall$ `x` $\in$ `X` we have that `Pr(Orc(x) = y) = 1/Sizeof(Y)` $\forall$ `y` $\in$ `Y.`
- A random oracle behaves like an "ideal" cryptographic hash function and as such is a valuable tool to prove security assertions under the so called *random oracle model*[1]. The previous point ensures that all the possible hash function outputs (*digests*) are *equiprobable*.
- The classical case where adversaries are bounded by computational factors is called the *standard model*.
- A cryptographic protocol that is proven secure in the random oracle model may result insecure when replacing the random oracle with a "real life" hash function as `SHA-1`, `SHA-256`, etc.

---

1. M. Bellare and P. Rogaway, *Random oracles are practical: A paradigm for designing efficient protocols*, In Proceedings of the 1st ACM Conference on Computer and Communications Security (1993), 62 -73.

Encryption, Decryption and Signing Oracles

- An encryption/decryption/signing oracle is also an abstract entity that will perform the corresponding operations as an “on demand” service to all legitimate and illegitimate parties.
- This entity will use (without disclosing them) the same keys as the legitimate key owners for both symmetric and asymmetric cryptosystems returning the authentic plaintext, ciphertexts or signed documents!
- Assuming `E` is a symmetric encryption primitive and `k` is the secret symmetric key, the encryption oracle will return $y = E_k(x)$ for any given input plaintext `x` and the decryption oracle will return `x` such as $E_k(x) = y$ for any given input ciphertext `y`.
- In Public Key Cryptosystems the oracle is only necessary for secret key operations (whether is signing or decrypting) since public key operations are openly available.
- As a result, assuming `E'` is a public key encryption system and `pubk` and `privk` the public and private keys, the decryption oracle will return `x` such as $E'_{pubk}(x) = y$ for any given input ciphertext `y`.
- Similarly, assuming `S` is a public key digital signature system and `pubk` and `privk` are respectively the validation and signing keys, the signing oracle will return `y` such as $S_{privk}(x) = y$ for any given input `x` (the data to be signed).
- *Chosen-plaintext* and *(adaptive) chosen-ciphertexts* attacks are normally modeled on the assumption that these oracles are available to the adversary.

Indistinguishable Ciphertexts and Semantic Security

- *Ciphertext Indistinguishability* is a property of cryptographic protocols that ensures that an adversary will be unable to distinguish between different ciphertexts for given plaintexts.
- Let's assume an adversary with polynomial time computing capability and unlimited access to an encryption oracle with secret key `k`, proceeds to the following steps:
  - He generates two plaintexts M_0 and M_1 and sends them to the encryption oracle.
  - The encryption oracle picks an index i at random $i \in \{0,1\}$ and sends the corresponding ciphertext $c_i = E_k(M_i)$ to the adversary.
  - The adversary is free to perform any additional computations including calls to the encryption oracle for any other M_j with $j \notin \{0,1\}$
- We say that a cryptosystem is *indistinguishable under chosen plaintext attacks* (***IND-CPA***) if such an adversary has a negligible advantage over random guessing to successfully picking the right index i (Prob = 1/2 + ε with ε small and bound).
- It should be noted that for public key cryptosystems the presence of an encryption oracle is unnecessary since encryptions involves public keys and consequently can be easily performed by the adversary.
- Cryptosystems featuring IND-CPA are also said to provide ***semantic security***[1].

1. S. Goldwasser, S. Micali. *Probabilistic encryption and how to play mental poker keeping secret all partial information*. Proc. 14th Symposium on Theory of Computing: 365–377. 1982.

Problems of Deterministic Encryption

- Cryptographic algorithms display a deterministic behaviour since encryption and decryption operations yield the same results over identical inputs.
- This may result in clear weaknesses in terms of *ciphertext indistinguishability* if an encryption oracle is available or if public key cryptosystems are used.
- As an example, let's assume Alice sends a simple message (i.e. 'yes' or 'no') to Bob encrypted with Bob's public key. If the adversary can guess the *semantics* of the message, he could easily compute the corresponding ciphertexts $C_{yes} = E_{Bobpubkey}(\text{'Yes'})$ and $C_{non} = E_{Bobpubkey}(\text{'Non'})$.
- If no extra random is added before encryption both ciphertexts are clearly distinguishable with Prob = 1 and, as a result, plaintexts would be disclosed without knowledge of Bob's private key.
- The adversary can even build a *codebook* of known plaintexts made of semantically sound messages and compare them with observed ciphertexts for eventual matches without cryptanalysing the algorithm!
- This problem also applies to symmetric cryptosystems provided that an encryption oracle is available and that no random Initialization Vector (IV) is used (more on this subject in the section describing block ciphers).

Probabilistic Encryption

- Consists in adding randomness to the cryptosystem by processing the plaintext *before* applying the encryption function. As a result, when dealing with multiple encryption instances the same plaintext will result in different ciphertexts.
- Building semantically sound codebooks becomes useless for adversaries. The final aim being to obtain *ciphertext indistinguishability* and *semantic security* for public-key cryptosystems.
- Initial probabilistic encryption approaches[1] had unpractical message expansion factors.
- The most commonly used solution is *Optimal Asymmetric Encryption Padding*[2] (OAEP) where the initial plaintext P is combined with a hashed random number R as follows:

  $M_1 := P \oplus h(R)$ and $M_2 := R \oplus h(M_1)$.

  M_1 and M_2 are then encrypted: $C_1 = E_{pubk}(M_1)$ and $C_2 = E_{pubk}(M_2)$ and sent.

  Upon decryption R is computed: $R = M_2 \oplus h(M_1)$ and then P: $P = h(R) \oplus M_1$.
- The security proofs provided in the initial OAEP publication have been questioned in recent papers[3]. OAEP is the basis of RSA-PKCS1 encryption standard.

---

1. S. Goldwasser, S. Micali. *Probabilistic encryption and how to play mental poker keeping secret all partial information*. Proc. 14th Symposium on Theory of Computing: 365–377. 1982.
2. M. Bellare, P. Rogaway. Optimal Asymmetric Encryption -- How to encrypt with RSA. Extended abstract in Advances in Cryptology - Eurocrypt '94 Proceedings, Lecture Notes in Computer Science Vol. 950, A. De Santis ed, Springer-Verlag, 1995.
3. D Brown, *Unprovable Security of RSA-OAEP in the Standard Model*, IACR eprint no 2006/223, http://eprint.iacr.org/2006/223. 2006

Systèmes de Cryptage Historiques

- Pendant des siècles la confidentialité a été la seule application de la cryptographie...
- I av. JC, *Caesar Cipher*: Cryptage à substitution mono-alphabétique

  $e_k(x) = (x + k) \bmod 26,\ d_k(y) = (y - k) \bmod 26$

  $x, y, k \in Z_{26}$

  - Exemple: E_1('bonjour') = 'cpokpws'
  - Cryptanalyse: facile basée sur la fréquence des caractères
- XVI siècle, *Vigenère*: Cryptage à substitution polyalphabétique

  $e_k(x_1,...x_n) = (x_1+k_1,...,x_m+k_m,x_{m+1}+k_1,...x_n+k_i) \bmod 26$

  $d_k(y_1,...y_n) = (y_1-k_1,...,y_m-k_m,y_{m+1}-k_1,...y_n-k_i) \bmod 26$

  $(x_1,...,x_n,y_1,...y_n,k_1,...,k_m) \in Z_{26}$

  - Cryptanalyse: trouver la taille m de la clé en identifiant les portions de ciphertext répétées et ensuite analyser les blocs séparés comme dans le *Caesar Cipher*
- *Transposition Ciphers* (*Porta*, 1563): La clé définit une *permutation* sur le *plaintext.*
- Autres exemples: cf. [Sti95] et [Men97].
- A noter que ces techniques sont toujours à la base des systèmes de cryptage actuels.
- *W. Churchill* à propos des Services d'Intelligence Britanniques capables de cryptanalyser les systèmes de cryptage allemands (*Enigma*): "*that secret weapon that won the war*"

Le *One-Time Pad*

Soit $n \geq 1$ et les espaces P,C,K resp. des plaintexts, ciphertexts et clés possibles tels que: P, C, $K = (Z_2)^n$. Soient $x = (x_1,x_2,...,x_n) \in X$, $y = (y_1,y_2,...,y_n) \in Y$ et $k =(k_1,k_2,...,k_n) \in K$. Alors, les operations d'encryption et decryption d'un *one-time pad* (aussi appellé *Vernam Cipher*) sont définies comme suit:

$$E_k(x_i) = x_i \oplus k_i \qquad 1 \leq i \leq n$$

$$D_k(y_i) = y_i \oplus k_i \qquad 1 \leq i \leq n$$

- Si les k_i sont choisis de manière indépendante et aléatoire, le one-time pad est *unconditionellement sûr* contre des attaques *ciphertext-only* ce qui veut dire que le fait d'observer des ciphertexts n'est d'aucune aide pour la cryptanalyse, ou dans d'autres termes, que l'entropie de X n'est pas diminuée après l'obervation des ciphertexts, soit: $H(X|C) = H(X)$. ($H(X)$ = fonction d'entropie, $H(X|C)$ = entropie conditionnelle)
  Ceci reste vrai même si l'adversaire a des ressources de calcul infinies!
- Problème: Shannon a prouvé qu'une condition nécessaire pour qu'un système de cryptage à clé symétrique soit inconditionnellement sûr est que $H(K) \geq H(X)$. En admettant que les composants d'une clé de m bits sont aléatoires et équiprobables, on a que $H(K) = m$ et donc que $m \geq H(X)$. Donc pour satisfaire l'hypothèse de Shannon indépendamment de l'entropie de X, il faut que la longueur de la clé aléatoire soit au moins aussi grande que celle du plaintext!

One-Time Pad (II)

- Une première conséquence de cette propriété est que la clé ne peut (même partiellement) être réutilisée. Voyons le résultat de la réutilisation de la même clé sur deux plaintexts différents:

$$E_k(x_a) = y_a = x_a \oplus k$$

$$E_k(x_b) = y_b = x_b \oplus k$$

$$y_a \oplus y_b = x_a \oplus k \oplus x_b \oplus k = x_a \oplus x_b \text{ !!!}$$

  ce qui avec des plaintexts de faible entropie permet de retrouver les deux plaintexts et même de calculer la clé k car:

$$k = y_a \oplus x_a$$

- Ceci signifie que le one-time pad est vulnerable à l'attaque *known plaintext*, même si ceci n'a pas d'importance si une nouvelle clé est regénérée pour chaque nouveau message.
- La contrainte sur la longueur de la clé pose un problème évident dans la distribution et la gestion des clés. De ce fait l'utilisation réelle du one-time pad est très peu sinificative.
- L'avènement de la *cryptographie quantique* proposant des canaux confidentiels de distribution de clés de longueur ilimitée a rélancé l'intérêt du one-time pad mais l'application de ces techniques aux réseaux classiques des télécommunications est pour le moment impossible.

Stéganographie

- Au lieu de rendre le message intelligible comme la cryptographie, la **stéganographie** (*steganography*) cache un message à l'intérieur d'un autre à l'aide de techniques diverses. Les deux éléments constituants de toute solution stéganographique sont les suivants:
  - Un canal physique ou logique différent de celui qui, de toute évidence, transporte l'information (aussi appelé *canal subliminal*).
  - Une indication ou un mécanisme secret permettant d'identifier le canal et d'accéder aux informations qu'il transporte.
- Exemples des solutions stéganographiques classiques:
  - Former un message secret en utilisant les premières lettres de tous les mots d'un texte.
  - Utiliser une encre invisible pour cacher les parties secrètes d'un texte
- [Way93][1] propose une solution stéganographique novatrice: former un message secret en utilisant les *least significant bit*s des *frames* présents sur un *Kodak Photo CD*. Pour une image de 2048x3072 avec une profondeur RGB de 24 bits, ceci donne 2.3 Mb !, en utilisant seulement 1 des 24 bits (ce qui ne détériore pas la qualité de l'image originale).

1.[Way93]: Wayner, P. *Should Encryption be Regulated?* BYTE Review, 1993.

Symmetric Cryptography
Outlook

- *Stream ciphers*
  - *Synchronous and asynchronous stream ciphers*
  - *Linear Feedback Shift Registers (LFSR)*
  - The **RC4** algorithm
- *Block ciphers*
  - Modes of operation: ***ECB, CBC, CFB, OFB*** and ***CTR***
  - *Product Ciphers* and *Feistel ciphers*
  - **DES**, **2DES** and **3DES**
  - **AES**
  - *Differential* and *Linear Cryptanalysis*

Cryptage en chaîne (*Stream Ciphers*)

- Les *stream ciphers* constituent une famille de systèmes de cryptage où la taille du bloc encrypté est égale à 1 bit.
- Les *stream ciphers* sont généralement composés de deux phases:
  - Une phase de ***génération*** de la séquence d'éléments formant la clé (le *keystream*).
  - Une phase de ***substitution*** où les bits du plaintext subissent une opération spécifique dépendante du *keystream*.
- Un exemple évident d'un *stream cipher* est le ***one-time pad*** avec:
  - Une phase de génération du *keystream* effectuée par un générateur (*pséudo*-) aléatoire.
  - Une phase de substitution qui consiste à effectuer un **xor** (⊕) avec le *keystream*.

Stream Ciphers: Caractéristiques

- *Rapidité*: Le cryptage se fait directement au niveau des registres. Idéal pour des applications nécessitant un cryptage "*on the fly*" comme le *video streaming*.
- *Facilité*: Les opérations peuvent être effectuées par des systèmes ayant des ressources CPU limitées.
- *Pas* (ou peu...) *besoin de mémoire/buffering*.
- *Propagation des erreurs limitée ou absente:* la retransmission des paquets fautifs suffit normalement (adapté aux applications où les pertes de paquets sont fréquentes comme les transmissions sans fil (*WiFi*)).
- Inconvénients:
  - La qualité en termes de *randomness* du *keystream* généré détermine la robustesse du système.
  - La *réutilisation* du *keystream* permet une cryptanalyse facile (cf. le *one-time pad*).

Stream Ciphers Synchrones

- Le *keystream* généré dépend seulement de la clé et non pas du *plaintext* ni du *ciphertext*.
- Le processus d'encryption d'un *stream cipher synchrone* est décrit par les équations suivantes:

  $\sigma_{i+1} = f(\sigma_i, k)$ $\quad z_i = g(\sigma_i, k)$ $\quad c_i = h(z_i, m_i)$

  avec σ_i l'état initial qui peut dépendre de la clé **k**, **f** la fonction qui détermine l'état suivant, **g** la fonction qui produit le *keystream* $\mathbf{z_i}$ et **h** la fonction de sortie qui produit le ciphertext $\mathbf{c_i}$ à partir du plaintext $\mathbf{m_i}$.

- Schématiquement, le mode de fonctionnement d'un *stream cipher synchrone* est le suivant:

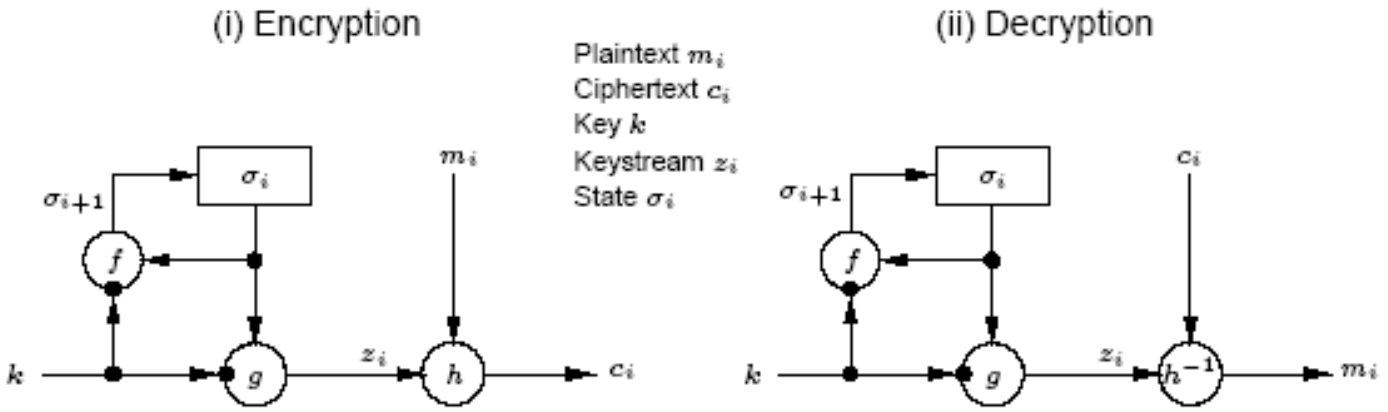
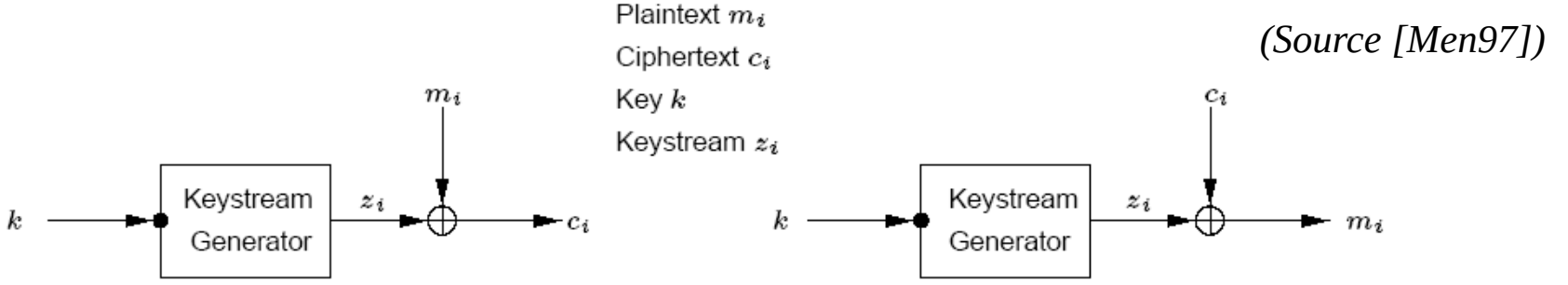


Figure 6.1: *General model of a synchronous stream cipher.*

(Source [Men97])

Stream Ciphers Synchrones: Caractéristiques

- Nécessitent la **synchronisation** de l'émetteur et du récepteur: En plus d'utiliser la même clé **k**, les deux doivent se trouver dans le même état pour que le processus fonctionne. Si la synchronisation est perdue il faut des mécanismes externes pour la récupérer (*marqueurs spéciaux, analyses de redondance du plaintext*, etc.)
- **Pas de propagation d'erreur**. La modification du ciphertext pendant la transmission n'entraîne pas des perturbations dans des séquences de ciphertext ultérieures (cependant, la suppression d'un ciphertext provoquerait la désynchronisation du récepteur).
- **Attaques actives**: l´insertion, l'élimination ou le *replay* de parties de ciphertext sont détectés par le récepteur. Cependant, un adversaire pourrait modifier certains bits du ciphertext et analyser l'impact sur le plaintext correspondant. Des mécanismes d'authentification d'origine supplémentaires sont nécessaires afin de détecter ces attaques.
- Cas les plus fréquent des *Stream Cipher Synchrones*: le **stream cipher additif** (cf. *le one-time pad*) où les fonctions **f** et **g** générant le keystream sont remplacées par un générateur aléatoire et la fonction **h** est une addition modulo 2 (xor):



(Source [Men97])

Stream Ciphers Asynchrones

- Aussi appelés **auto-synchronisés** (***self synchronizing*** *ciphers).*
- Le keystream généré dépend de la clé ainsi que d'*un nombre fixé de ciphertexts* précédents.
- Le processus d'encryption d'un *stream cipher asynchrone* est décrit par les équations suivantes:

$\sigma_i = (c_{i-t}, c_{i-t+1}, \dots, c_{i-1})$ $\quad z_i = g(\sigma_i, k)$ $\quad c_i = h(z_i, m_i)$

avec σ_i, **g** et **h** comme pour le cas synchrone.

- Schématiquement, le mode de fonctionnement d'un *stream cipher asynchrone* est le suivant:

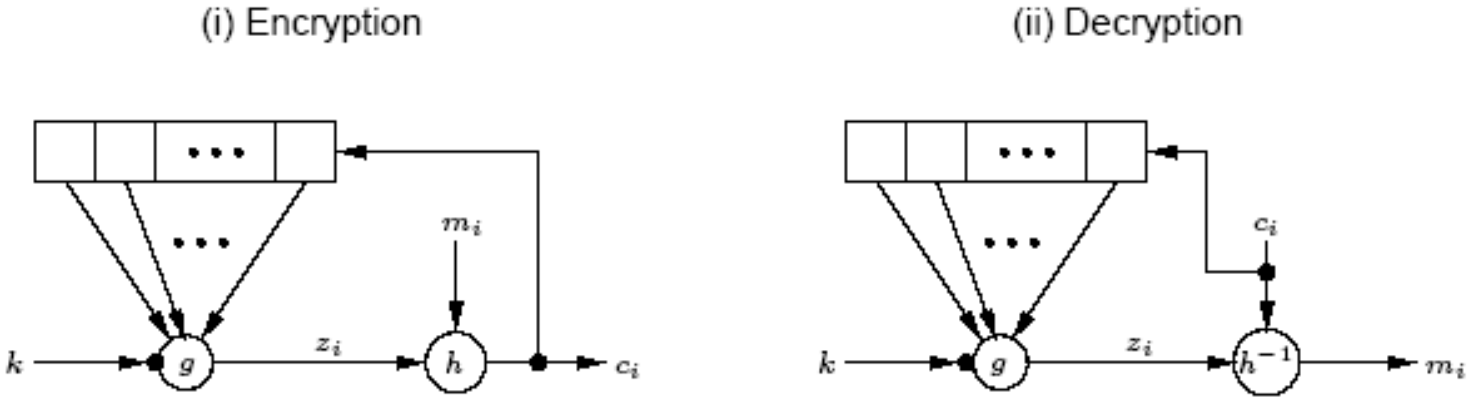


Figure 6.3: *General model of a self-synchronizing stream cipher.*

(Source [Men97])

Stream Ciphers Asynchrones: Caractéristiques

- **Auto-synchronisation**: En cas d'élimination ou d'insertion de ciphertexts en cours de route, le récepteur est capable de se *re-synchroniser* avec l'émetteur grâce à la mémorisation (*buffer*) d'un nombre de ciphertext précédents.
- **Propagation d'erreurs limitée**: La propagation d'erreurs s'étend uniquement au nombre de bits du ciphertext mémorisés (taille du *buffer*). Après, la decryption se déroule à nouveau correctement.
- **Attaques actives**: La modification de fragments du ciphertext sera plus facilement détecté que dans le cas synchrone à cause de la propagation d'erreurs. Cependant, comme le récepteur est capable de s'auto-synchroniser avec l'émetteur, même si des ciphertexts sont éliminés ou insérés en cours de route, il convient de vérifier l'intégrité et l'authenticité du flot entier.
- **Diffusion des statistiques du plaintext**: Le fait que chaque bit du plaintext aura une influence sur la totalité des ciphertexts subséquents se traduit par une plus grande dispersion des statistiques du plaintext comparée au cas synchrone...
- ... Il convient, donc, d'**utiliser des stream ciphers asynchrones lorsque l'entropie des plaintexts est limitée** et pourrait permettre des attaques ciblées aux plaintexts fortement redondants.

Stream Ciphers: Générateurs de Keystreams

- Lorsqu'il convient de générer un keystream d'une longueur **m** à partir d'une clé secrète de longueur **l** avec **l** << **m**, on fait appel à des générateurs de keystreams.
- Le plus courant de ces générateurs est le *Linear Feedback Shift Register* (LSFR).
- Un LSFR a les caractéristiques suivantes:
  - S'adapte très bien aux implantations hardware.
  - Produit des séquences de périodes longues et avec une qualité aléatoire notable (*randomness* assez forte)
  - Se base sur les propriétés algébriques des combinaisons linéaires.
- Exemple générique d'un **LSFR** de longueur **L**:

Figure 6.4 depicts an LFSR. Referring to the figure, each c_i is either 0 or 1; the closed semi-circles are AND gates; and the feedback bit s_j is the modulo 2 sum of the contents of those stages i, $0 \leq i \leq L-1$, for which $c_{L-i} = 1$.

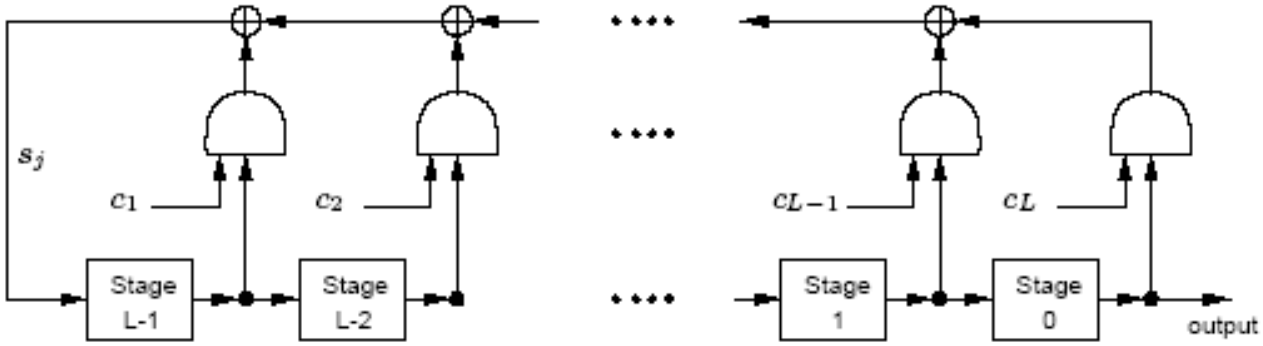
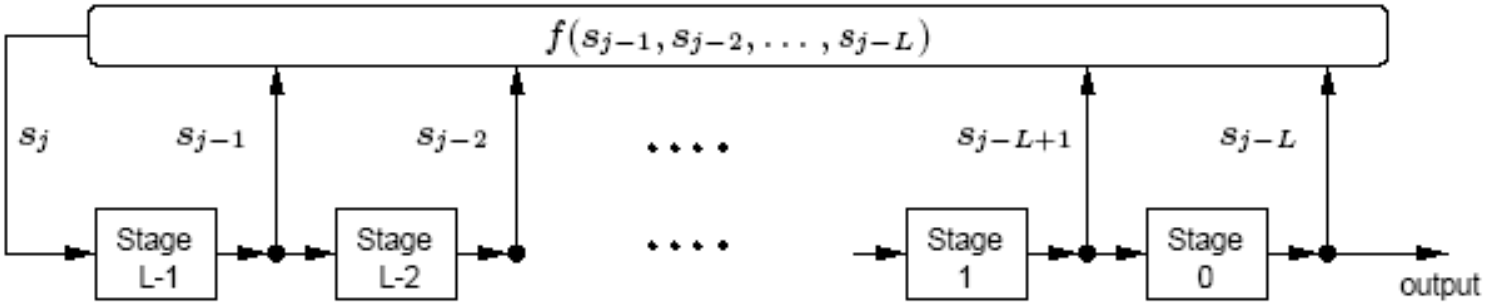


Figure 6.4: *A linear feedback shift register (LFSR) of length L.*

(Source [Men97])

LSFRs: Quelques Remarques

- Les LSFRs sont des constructions très répandues dans la cryptographie et dans la théorie de codes.
- Un grand nombre de *stream ciphers basés sur les LSFRs* (surtout dans la sphère militaire) ont été développés dans le passé.
- Malheureusement, le niveau de sécurité offert par ces systèmes est jugé insuffisant de nos jours (comparé à celui des *blocks ciphers*...)
- La métrique permettant d'analyser un LFSR est sa *complexité linéaire* (**linear complexity**). L'algorithme de *Berlekamp-Massey*[1] permet de déterminer la complexité linéaire d'un LSFR et de calculer ainsi un nombre arbitrairement grand de séquences générées par un LSFR.
- Une solution pour augmenter la complexité est de substituer la combinaison linéaire des bits du ciphertext par une fonction non linéaire f. Ce sont les ***Non Linear Feedback Shift Registers***:



1. J.L. Massey. *Shift Register Synthesis and BCH Decoding*. IEEE Transactions on Information Theory, 15 (1969), 122-127.

Software Cipher Streams: RC4

- Le grand désavantage des stream ciphers basés sur des registres est qu'ils sont très lents en version programmée dans une machine générique. $RC4_{TM}$ est un *stream cipher à clé variable* développé en 1987 par *Ron Rivest* pour la société *RSA security*. Il est très rapide (10 fois plus rapide que DES!)
- Pendant 7 ans, cet algorithme était breveté et les détails son fonctionnement interne était dévoilés seulement après la signature d'un contrat de confidentialité. Depuis sa publication (non officielle) dans un *newsgroup* en 1994, il est globalement discuté et analysé dans toute la communauté cryptographique.
- L'algorithme travaille en mode *synchrone* (le *keystream* est indépendant du ciphertext et du plaintext).
- Il est composé de combinaisons linéaires et non linéaires. L'élément clé est une boîte de substitution (**S-box**) de taille 8x8 dont les entrées sont une permutation des chiffres 0 à 255. La permutation est une fonction de la clé principale de taille variable avec $0 < \text{len}(k) \leq 255$. L'encryption finale est obtenue par un **xor** entre le *keystream* et le *plaintext*.
- RC4 est utilisé dans un grand nombre d'applications commerciales: *Lotus Notes, Oracle SQL, MS Windows, SSL, etc.* Il est l'objet d'un grand nombre de travaux[1] analytiques et exhaustifs qui ont réussi à compromettre la sécurité du *key scheduling* et du *PRGA*.
- En particulier l'application de RC4 sur les *Wired Equivalent Privacy* (***WiFi WEP***) protocole a été "cassée" suite à une faille dans le mode d'utilisation du protocole.

1. http://www.wisdom.weizmann.ac.il/~itsik/RC4/rc4.html

RC4: Fonctionnement

- L'algorithme est constitué de deux étapes:
  - Le ***Key Scheduling Algorithm*** (**KSA**): Responsable de la permutation initiale qui remplira la *S-box* en fonction de la clé de longueur variable `len(k)` = ℓ.
  - Le **Pseudo Random Generator Algorithm** (**PRGA**): Génère le *keystream* de taille arbitraire en s'appuyant sur la *S-box*.

KSA(K)
Initialization:
$S \leftarrow \langle 0, 1, \ldots, N-1 \rangle$
$j \leftarrow 0$
Scrambling:
For $i \leftarrow 0 \ldots N-1$
$j \leftarrow j + S[i] + K[i \bmod \ell]$
$S[i] \leftrightarrow S[j]$

PRGA(S)
Initialization:
$i \leftarrow 0$
$j \leftarrow 0$
Generation loop:
$i \leftarrow i + 1$
$j \leftarrow j + S[i]$
$S[i] \leftrightarrow S[j]$
$t \leftarrow S[i] + S[j]$
Output $z \leftarrow S[t]$

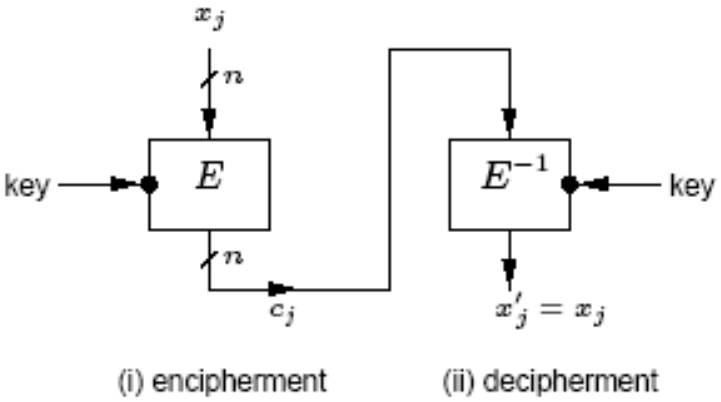
Source: Fluhrer, Mantin and Shamir. ***Attacks on RC4 and WEP***, Cryptobytes 2002

Cryptage par Blocs (*Block Ciphers*)

- Les *block ciphers* symétriques constituent la pierre angulaire de la cryptographie. Leur fonctionnalité principale est la confidentialité mais ils sont également à la base des services *d'authentification*, *fonctions de hachage*, *génération aléatoire*, etc.
- Définition: Un ***block cipher*** est une **fonction** qui fait correspondre à un bloc de n-bits un autre bloc de la même taille. La fonction est paramètrée par une clé K de k-bits. Afin de permettre une decryption unique, la fonction doit être **bijective**. *Chaque clé définit une bijection différente*. La taille d'entrée du bloc sur lequel s'applique l'encryption s'appelle aussi **taille nominale** de l'algorithme.
- Critères pour évaluer la qualité d'un *block cipher*:
  - **Taille/Entropie de la clé**: Idéalement, les clés sont équiprobables et l'espace des clés a une entropie égale à k. Une forte entropie de la clé protège des attaques *brute-force* à partir de *chosen/known plaintexts*. Les *block ciphers* modernes doivent avoir des clés d'au moins 128 bits.
  - **Performances**
  - **Taille du bloc**: Un bloc trop petit permettrait des attaques où des "dictionnaires" plaintext/ciphertext seraient construits. De nos jours, des blocs de taille ≥ 128 bits deviennent courants.
  - **Résistance cryptographique**: Le *block cipher* doit se montrer résistant à des techniques de cryptanalyse connues: *cryptanalyse linéaire ou différentielle, meet in the middle*, etc. L'effort inhérent à ces attaques (complexité, stockage, parallélisation, etc.) doit être équivalent à celui d'une attaque *brute force*.

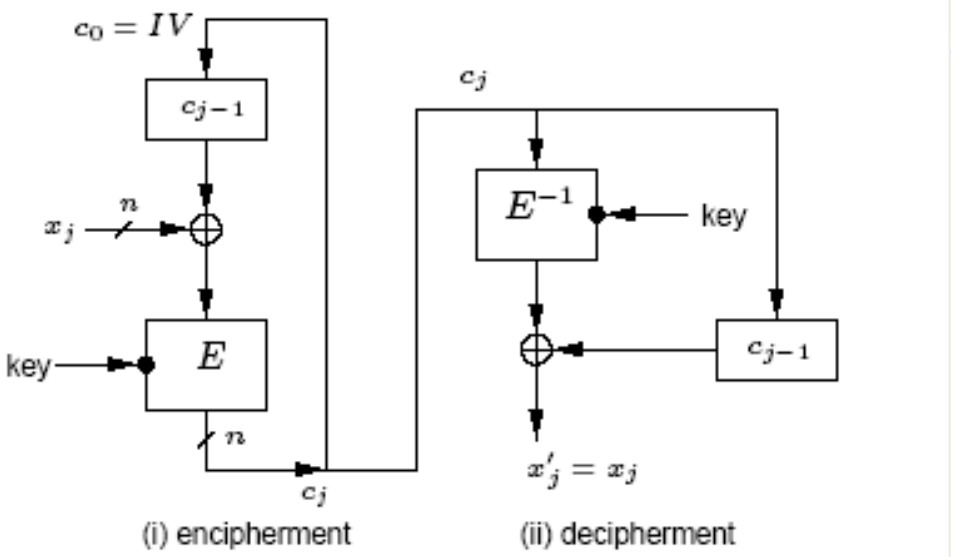
Block Ciphers: Modes d'Opération

Electronic Codebook (ECB)



- Des plaintexts identiques donnent lieu à des ciphertexts identiques
- Pas de propagation d'erreurs sur les blocs adjacents

Cipher-block Chaining(CBC)

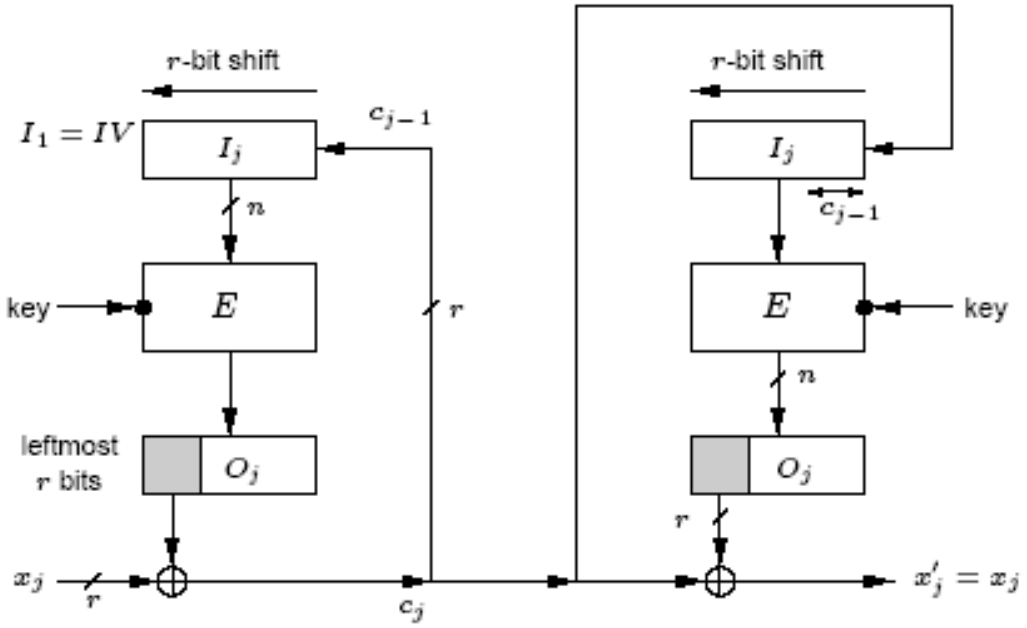


- Des plaintext identiques donnent lieu à des ciphertexts différents si le ***IV*** change
- Les *patterns* du plaintext sont "effacées" du ciphertext (chaînage)
- Une erreur sur c_j affecte uniquement la décryption des blocs c_j et c_{j+1}

Block Ciphers: Modes d'Opération (II)

Cipher Feedback Mode*: *CFB (Source [Men97])

c) Cipher feedback (CFB), r-bit characters/r-bit feedback



7.17 Algorithm CFB mode of operation (CFB-r)

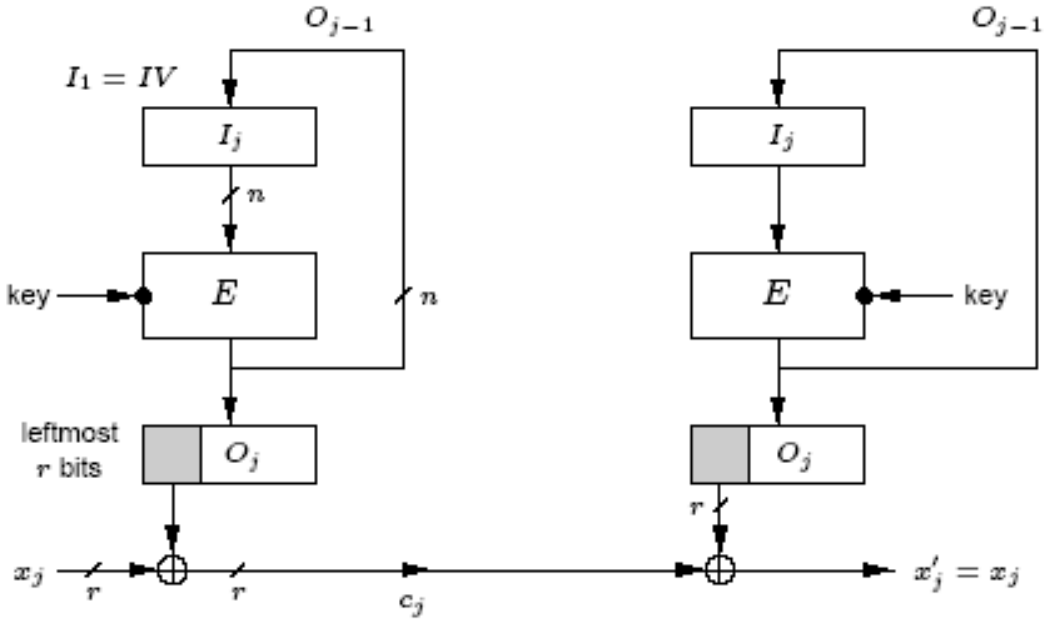
INPUT: k-bit key K; n-bit IV; r-bit plaintext blocks $x_1, \ldots, x_u$ $(1 \leq r \leq n)$.
SUMMARY: produce r-bit ciphertext blocks $c_1, \ldots, c_u$; decrypt to recover plaintext.

1. Encryption: $I_1 \leftarrow IV$. (I_j is the input value in a shift register.) For $1 \leq j \leq u$:
   (a) $O_j \leftarrow E_K(I_j)$. (Compute the block cipher output.)
   (b) $t_j \leftarrow$ the r leftmost bits of O_j. (Assume the leftmost is identified as bit 1.)
   (c) $c_j \leftarrow x_j \oplus t_j$. (Transmit the r-bit ciphertext block c_j.)
   (d) $I_{j+1} \leftarrow 2^r \cdot I_j + c_j \bmod 2^n$. (Shift c_j into right end of shift register.)
2. Decryption: $I_1 \leftarrow IV$. For $1 \leq j \leq u$, upon receiving c_j:
   $x_j \leftarrow c_j \oplus t_j$, where t_j, O_j and I_j are computed as above.

Block Ciphers: Modes d'Opération (III)

Output Feedback Mode: *OFB* (Source [Men97])

d) Output feedback (OFB), r-bit characters/n-bit feedback



(i) encipherment (ii) decipherment

7.20 Algorithm OFB mode with full feedback (per ISO 10116)

INPUT: k-bit key K; n-bit IV; r-bit plaintext blocks $x_1, \ldots, x_u$ $(1 \leq r \leq n)$.
SUMMARY: produce r-bit ciphertext blocks $c_1, \ldots, c_u$; decrypt to recover plaintext.

1. Encryption: $I_1 \leftarrow IV$. For $1 \leq j \leq u$, given plaintext block x_j:
   (a) $O_j \leftarrow E_K(I_j)$. (Compute the block cipher output.)
   (b) $t_j \leftarrow$ the r leftmost bits of O_j. (Assume the leftmost is identified as bit 1.)
   (c) $c_j \leftarrow x_j \oplus t_j$. (Transmit the r-bit ciphertext block c_j.)
   (d) $I_{j+1} \leftarrow O_j$. (Update the block cipher input for the next block.)
2. Decryption: $I_1 \leftarrow IV$. For $1 \leq j \leq u$, upon receiving c_j:
   $x_j \leftarrow c_j \oplus t_j$, where t_j, O_j, and I_j are computed as above.

Modes CFB et OFB: Caractéristiques

Les modes CFB et OFB fonctionnent comme un *stream cipher* avec un *keystream* généré par le bloc de cryptage. Dans CFB, le *keystream* dépend des ciphertexts précédents (asynchrone) alors que dans OFB, le keystream est entièment déterminé par la clé et le *IV* (synchrone).

Particularités de CFB:

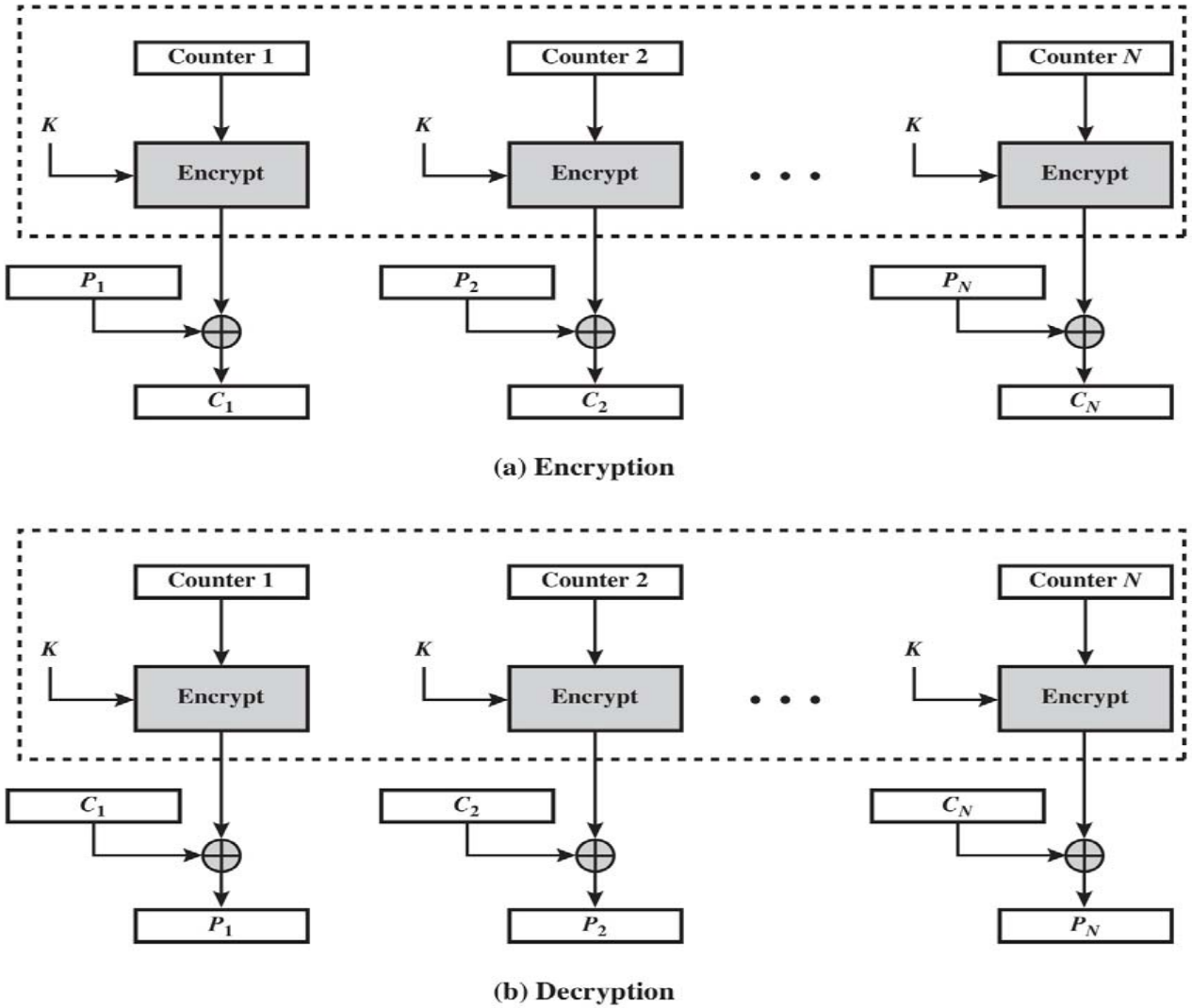
- Comme dans le mode CBC, des plaintext identiques sont traduits en ciphertexts différents si le IV change. Le IV n'est pas nécessairement confidentiel et peut être échangé en clair entre les parties.
- Le chaînage introduit également des dépendances entre les ciphertexts courants et les ciphertexts précédents. En particulier, si **n** est la taille nominal de l'algorithme et **r** est la taille des plaintexts, le ciphertext courant dépendra des **n/r** ciphertexts précédents (chaque itération décalera l'entrée fautive de **r** positions, après **n/r** itérations le ciphertext fautif sera "expulsé" complètement).
- La propagation d'erreurs obéit au même principe: une erreur dans un ciphertext se traduira par une mauvaise decryption des **n/r** ciphertexts suivants.

Particularités de OFB:

- OFB a un comportement identique aux modes CBC et CFB pour l'encryption de plaintext identiques.
- Pas de propagation d'erreurs sur les ciphertexts adjacents.
- Modifiez le IV si la clé ne change pas pour éviter la réutilisation du *keystream*!!!

Counter Mode (CTR Mode)

Fréquemment utilisé comme support d'encryption dans des protocoles de transfert de données comme ATM (*Asynchronous Transfer Mode*) et IPsec (*IP security*)



Source [Sta10]

Counter Mode (II)

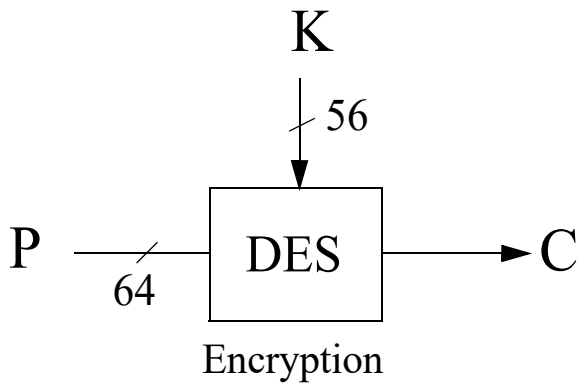
- Le *keystream* est généré par l'encryption d'un compteur aléatoire de taille 2^b (avec b la taille du bloc) et nécessaire pour la décryption. Ce compteur est incrémenté modulo 2^b après chaque itération.
- Travaille en mode synchrone. La réutilisation d'un même compteur se traduit par un *keystream identique*!
- Solution: Toujours incrémenter le compteur pour chaque flot encrypté de telle sorte que le compteur du premier bloc d'un flot soit plus grand que le dernier bloc du flot précédent.
- *Facilement parallélisable*: Le keystream peut être pré-calculé aussi bien pour l'encryption que pour la décryption. Profite pleinement des architectures SIMD car contrairement aux autres modes de chaînage il n'y a pas des dépendances entre les opérations des différents blocs.
- *Accès aléatoire à l'encryption/décryption de chaque bloc*: Contrairement aux autres modes de chaînage où la *i-ème opération dépend de la (i-1)-ème opération*.
- Si à ceci on ajoute l'*absence de propagation d'erreurs*, le mode compteur facilite la (re)transmission sélective des blocs de ciphertext, ce qui le rend très attractif pour la sécurisation de lignes à haut débit ainsi que pour les transferts encryptés de grands volumes d'information (p.ex. vidéo).

Product Ciphers et *Feistel Ciphers*

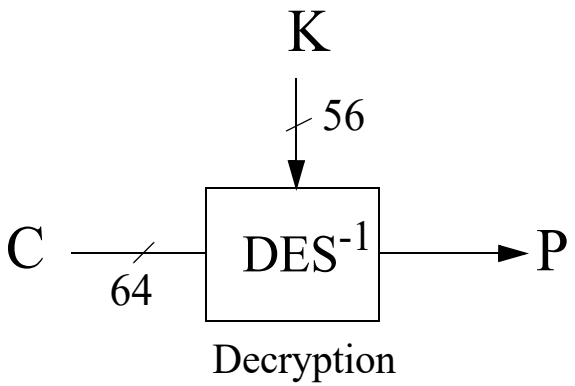
- Un ***product cipher*** est un schéma de cryptage combinant une série de transformations successives dans le but de renforcer la résistance à la cryptanalyse. Des transformations courantes pour un *product cipher* sont: des transpositions, des substitutions, des XORs, des combinaisons linéaires, des multiplications modulaires, etc.
- Un ***Feistel cipher*** est un *product cipher* itératif capable de transformer un plaintext de **2t** bits de la forme $(\mathbf{L_0}, \mathbf{R_0})$ composé par deux sous-blocs $\mathbf{L_0}$ et $\mathbf{R_0}$ de **t** bits en un ciphertext de taille **2t** de la forme $(\mathbf{R_r}, \mathbf{L_r})$ après **r** étapes (*rounds*) successives avec $\mathbf{r} \geq 1$. Chaque étape définit une *bijection* (inversible!) pour permettre une decryption unique.
- Des *permutations* et des *substitutions* sont les opérations les plus fréquentes.
- Les étapes $\mathbf{1} \leq \mathbf{i} \leq \mathbf{r}$ s'écrivent: $(\mathbf{L_{i-1}}, \mathbf{R_{i-1}}) \xrightarrow{\mathbf{K_i}} (\mathbf{L_i}, \mathbf{R_i})$ avec $\mathbf{L_i} = \mathbf{R_{i-1}}$ et $\mathbf{R_i} = \mathbf{L_{i-1}} \oplus f(\mathbf{R_{i-1}}, \mathbf{K_i})$. Les $\mathbf{K_i}$ sont des sous-clés, différentes pour chaque étape, générées à partir de la clé principale **K** du schéma de cryptage.
- Le nombre d'étapes propres à un *Feistel cipher* est normalement pair et $\geq \mathbf{3}$ (p.ex. DES a 16 étapes)
- Après l'exécution de toutes les étapes, un *Feistel cipher* effectue une permutation des deux parties $(\mathbf{L_r}, \mathbf{R_r})$ en $(\mathbf{R_r}, \mathbf{L_r})$.
- La decryption d'un *Feistel Cipher* est identique à l'encryption sauf que les sous-clés $\mathbf{K_i}$ sont appliquées en ordre inverse (De $\mathbf{K_r}$ à $\mathbf{K_1}$).

Data Encryption Standard (**DES**)

- DES a été l'algorithme cryptographique le plus important jusqu'à l'avènement d'AES en 2001.
- DES est un *Feistel Cipher* avec des blocs de 64 bits (taille nominale).
- La taille effective de la clé est de 56 bits (Un total de 64 bits avec 8 bits de parité).
- L'algorithme est constitué de **16** étapes avec **16** sous-clés de 48 bits générées (une clé par étape).
- DES (comme tout autre *block cipher*) peut être utilisé dans les 4 modes: ECB, CBC, CFB et OFB.
- Le schéma de fonctionnement de DES est le suivant:

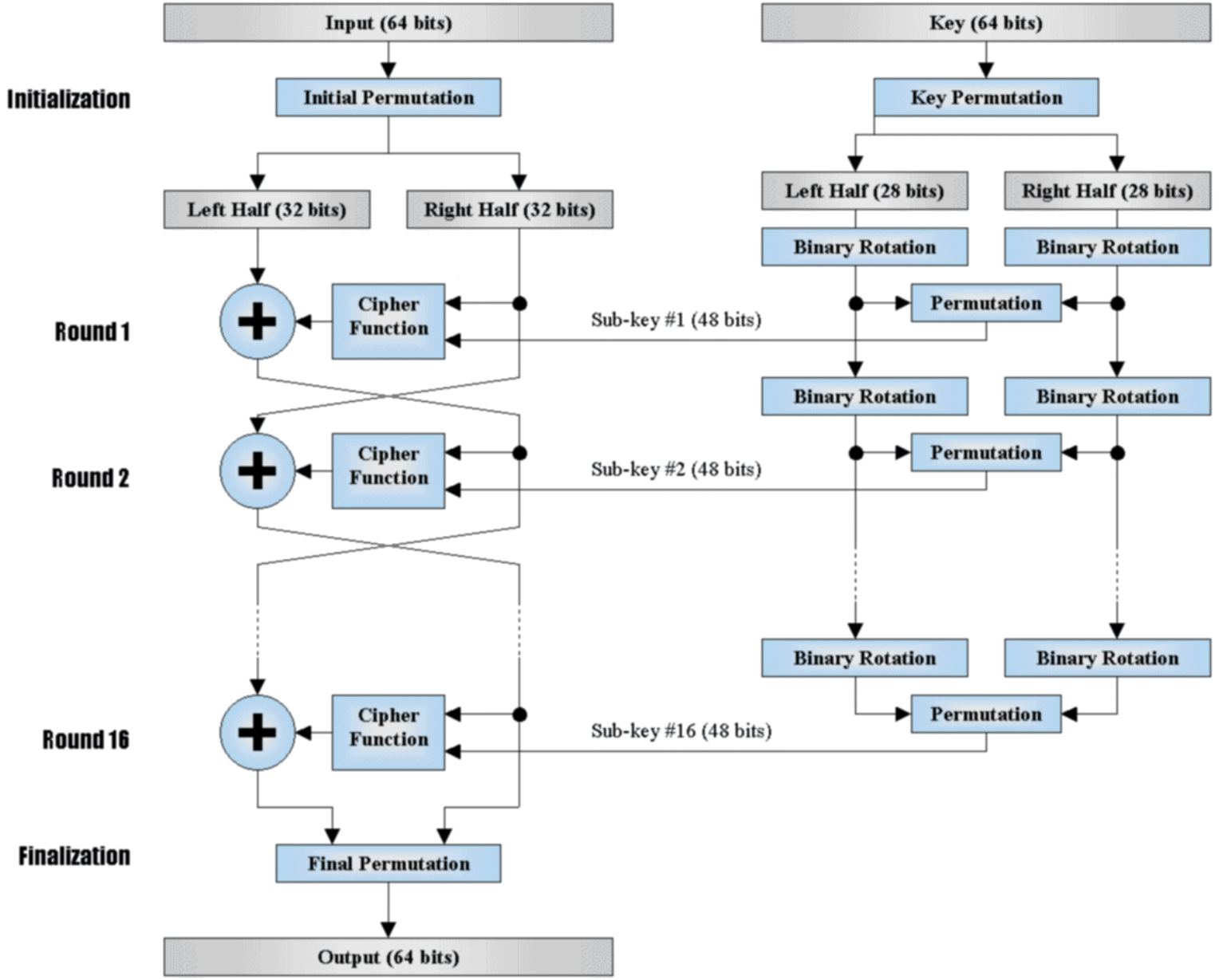


Encryption

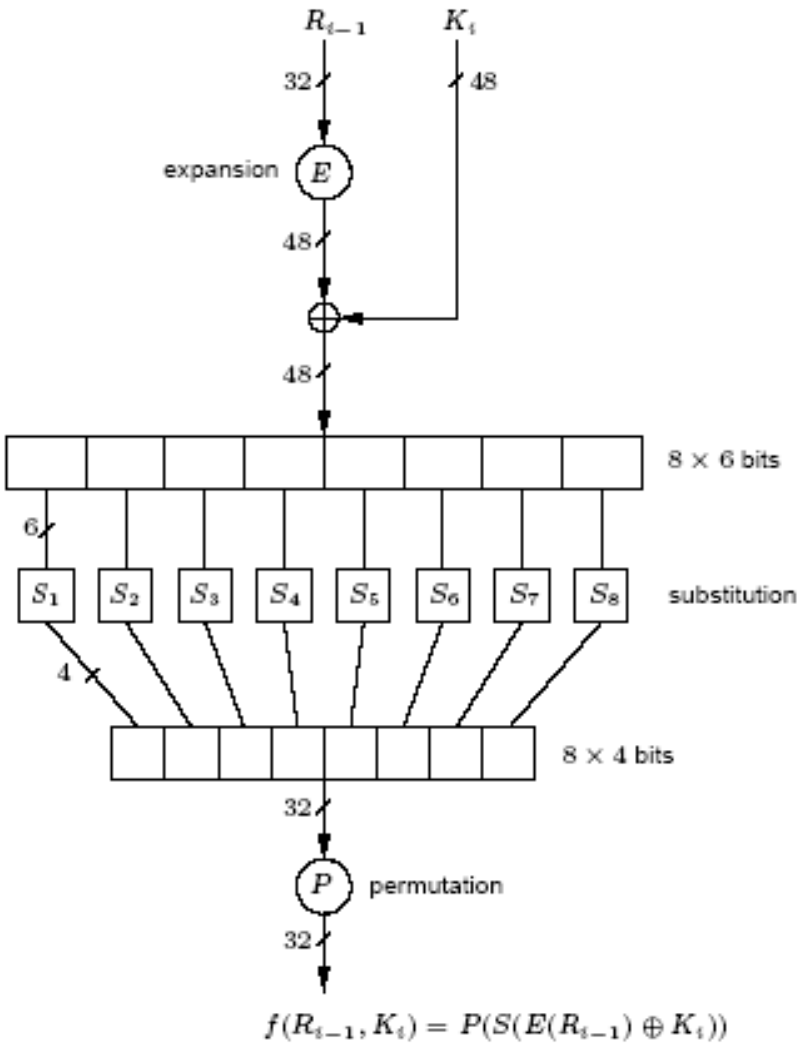


Decryption

DES: Schéma de Fonctionnement



DES: *Cipher Function*



(Source [Men97])

Figure 7.10: *DES inner function* f.

DES: Tables

IP							
58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

IP^{-1}							
40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

Table 7.2: *DES initial permutation and inverse (IP and IP^{-1}).*

E					
32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

P			
16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	3	9
19	13	30	6
22	11	4	25

(Source [Men97])

Table 7.3: *DES per-round functions: expansion E and permutation P.*

DES: *S-boxes*

S-Box 1: Substitution Box 1

Row / Column	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

S-Box 2: Substitution Box 2

Row / Column	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
1	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
2	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
3	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

DES: Fonctionnement

Cipher Fonction

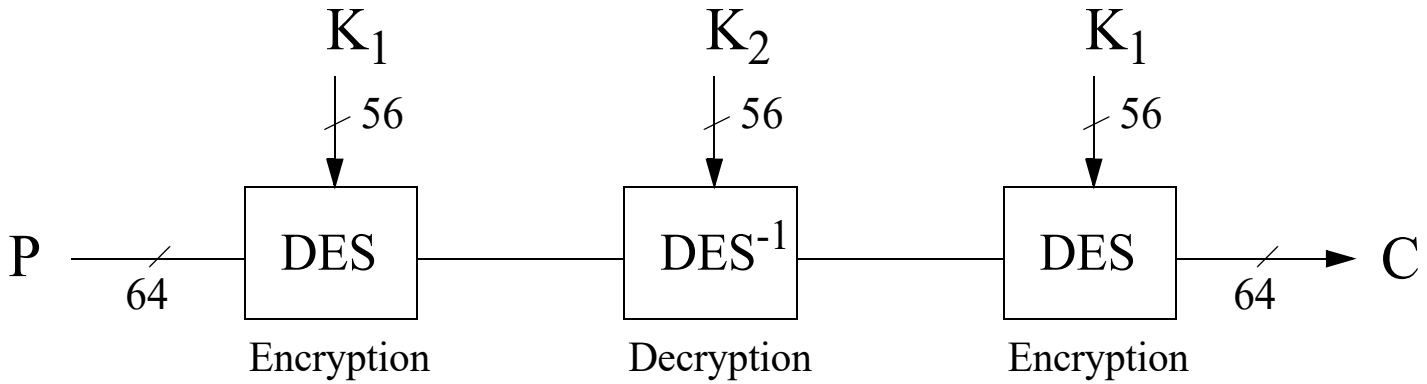
- **Expansion E**: Les 32 bits de l'entrée sont transformés en un vecteur de 48 bits en utilisant la table **E** (page 78). La première ligne de cette table indique comment sera généré le premier sous-bloc de 6 bits: on prendra en premier le 32^e bit et après les bits 1,2,3,4,5. Le deuxième sous-bloc commence par le 4^e bit ensuite les bits 5,6,7,8,9 et ainsi de suite...
- ***Key addition***: XOR du vecteur de 48 bits avec la clé.
- **S-boxes**: On applique **8 S-boxes** sur le vecteur de 48 bits résultant du XOR précédent. Chacune de ces S-boxes prend un sous-bloc de 6 bits et le transforme en un sous-bloc de 4 bits. Les S-boxes 1 et 2 sont présentées en page 79. L'opération s'effectue de la manière suivante: Si on dénote les 6 bits d'input de la S-box comme: $a_1a_2a_3a_4a_5a_6$. La sortie est donnée par le contenu de la cellule située dans la ligne $a_1 + 2a_6$ et la colonne $a_2 + 2a_3 + 4a_4 + 8a_5$.
- **Permutation P**: La permutation P (page 78) fonctionne comme suit: Le premier bit est envoyé à la 16^e position, le deuxième à la 7^e position et ainsi de suite.

Permutations IP et IP^{-1}

- Agissent respectivement au début et à la fin du traitement du bloc et sur l'ensemble des 64 bits (voir les tables en page 78 pour les détails).

DES et Triple-DES

- La taille de l'ensemble de clés ($\{0,1\}^{56}$) constitue la plus grande menace qui pèse sur DES avec les ressources de calcul actuels. En 1999 il a suffit de 24 heures pour trouver la clé à partir d'un *known plaintext* en utilisant une technique brute force massivement parallèle (100'000 PCs connectés sur Internet)[1].
- **Triple DES** nous met à l'abri de ces attaques *brute force* en augmentant l'espace des clés possibles à $\{0,1\}^{112}$. Schématiquement, il fonctionne de la manière suivante:



- Cette alternative permet de continuer à utiliser les "boîtes" DES (hardware et software) en attendant une migration vers AES.
- Le niveau de sécurité obtenu par cette solution est très satisfaisant.
- L'impact en termes de performances de trois exécutions successives de DES reste un inconvénient pour certaines applications.

1. http://www.rsasecurity.com/company/news/releases/pr.asp?doc_id=462

DES: propriétés

- **DES n'est pas un groupe** (au sens algébrique) avec la composition: En d'autres termes, DES étant une permutation: $\{0,1\}^{64} \rightarrow \{0,1\}^{64}$, si DES était un groupe pour la composition, ceci voudrait dire que:

  $$\forall (K_1,K_2), \exists K_3 \text{ t.q. } E_{K2}(E_{K1}(x)) = E_{K3}(x)$$

  Cette propriété permet d'assurer que l'encryption composée (comme *Triple-DES*) augmente considérablement la sécurité de DES. Si DES était un groupe, la recherche exhaustive sur l'ensemble de clés possibles ($\{0,1\}^{56}$) permettrait de "casser" l'algorithme indépendamment du nombre d'exécutions consécutives de DES.
- **Clés faibles et mi-faibles** (***weak*** *and* ***semi-weak*** *keys*):
  - Une clé **K** est dite **faible** si $\mathbf{E_K(E_K(x)) = x}$.
  - Une paire de clés $\mathbf{(K_1, K_2)}$ est dite **mi-faible** si $\mathbf{E_{K1}(E_{K2}(x)) = x}$.
- Les clés faibles ont la particularité de générer de sous-clés identiques par paires ($k_1 = k_{16}$, $k_2 = k_{15}$,..., $k_8 = k_9$), ce qui facilite la cryptanalyse.
- DES a 4 clés faibles (et 6 paires de clés mi-faibles):

Clés faibles pour DES:

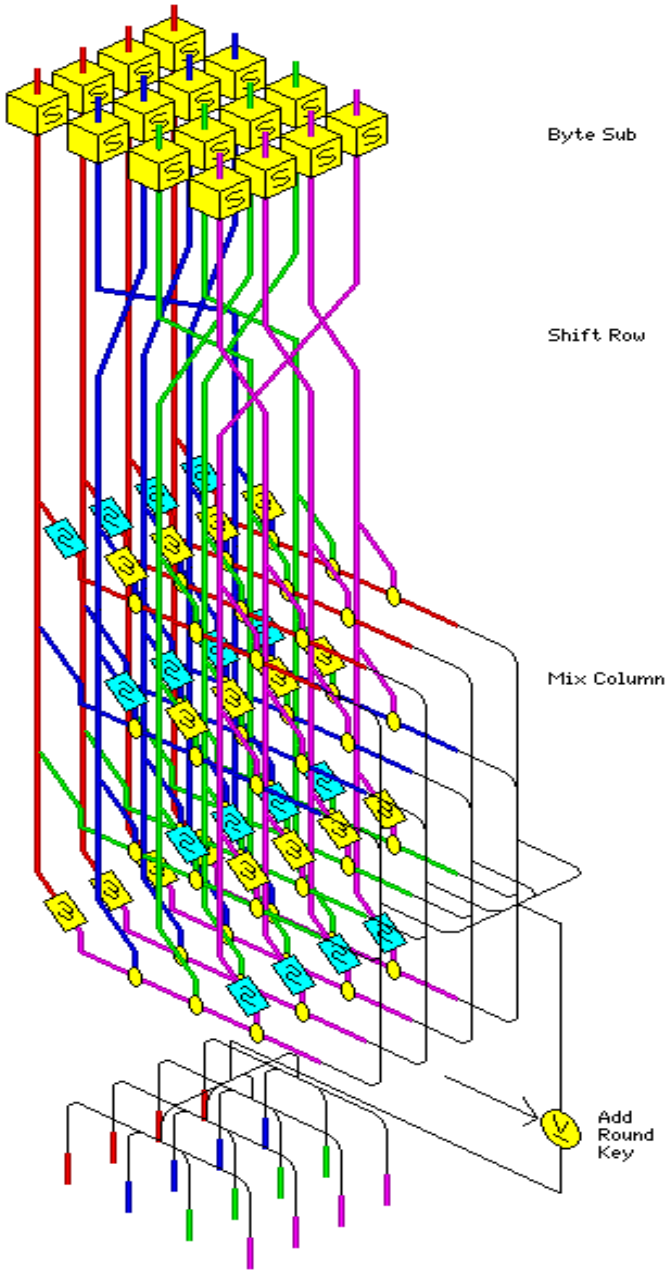
```
0101 0101 0101 0101
0101 0101 FEFE FEFE
FEFE FEFE FEFE FEFE
FEFE FEFE 0101 0101
```

Advanced Encryption Standard (AES)

- Adopté comme standard en Novembre 2001[1], conçu par *Johan Daemen* et *Vincent Rijmen*[2] *(*d'où son nom original *Rijndael).*
- Il s'agit également d'un *block cipher itératif* (comme DES) mais pas d'un *Feistel Cipher*
  - Blocs *Plaintext/Ciphertext*: 128 bits.
  - Clé de longueur variable: 128, 192, ou 256 bits.
- Contrairement à DES, AES est issu d'un processus de consultation et d'analyse ouvert à des experts mondiaux.
- Techniques semblables à DES (substitutions, permutations, XOR…) complémentées par des opérations algébriques simples et très performantes.
- Toutes les opérations s'effectuent dans le corps $GF(2^8)$: le corps fini de polynômes de degré ≤ 7 avec des coefficients dans GF(2).
- En particulier, un byte pour AES est un élément dans $GF(2^8)$ et les opérations sur les bytes (additions, multiplications,...) sont définies comme sur $GF(2^8)$.
- ~2 fois plus performant (en software) et $\sim 10^{22}$ fois (en théorie...) plus sûr que DES...
- Évolutif: La taille de la clé peut être augmentée si nécessaire.

1. http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf
2. http://csrc.nist.gov/CryptoToolkit/aes/rijndael/Rijndael-ammended.pdf

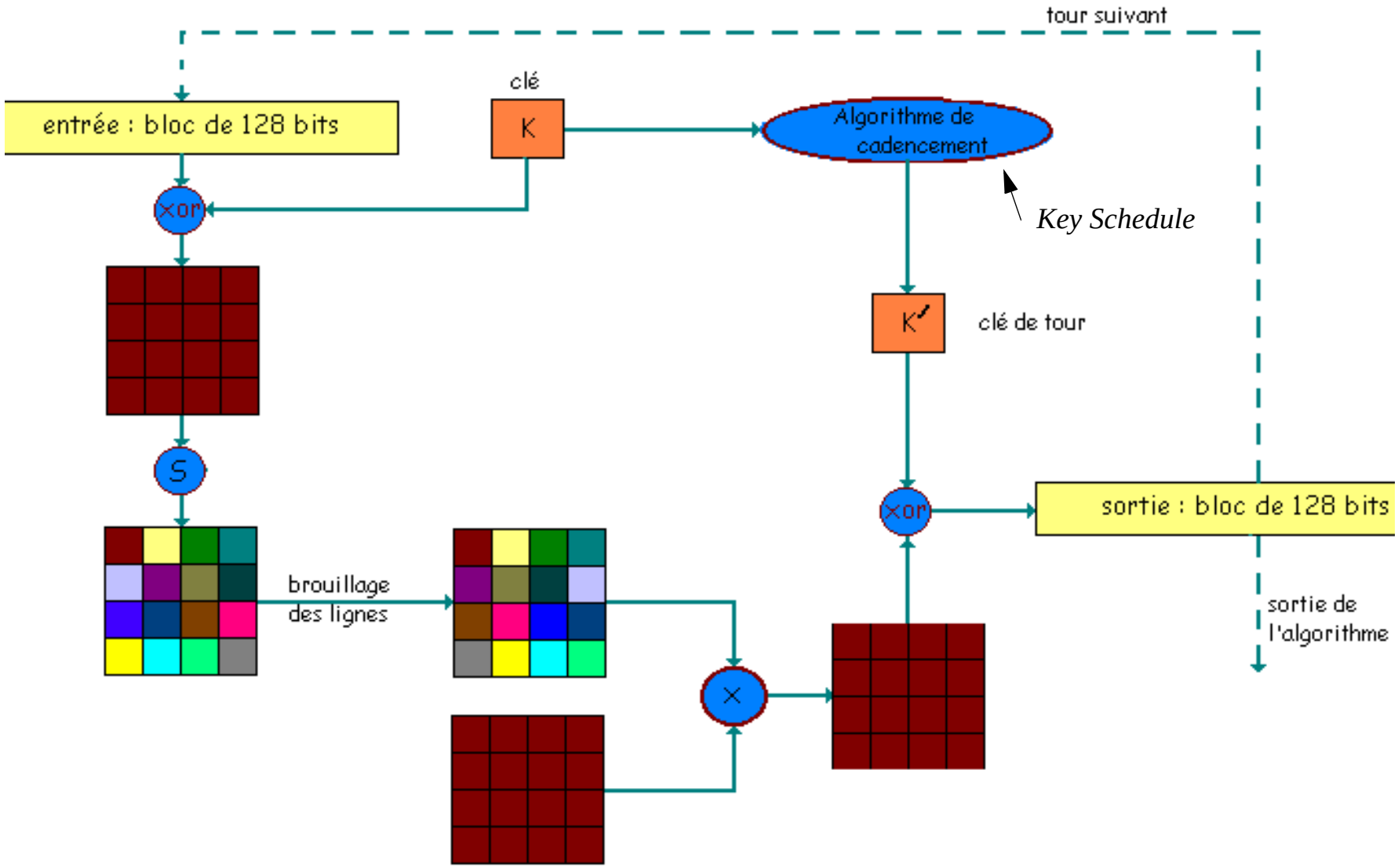
Détail d'une Etape (*round*) AES



L'unité de base sur laquelle s'appliquent les calculs est une matrice de 4 lignes et 4 colonnes (dans le cas d'une clé de 128 bits) dont les éléments sont des bytes:

- ***ByteSub***: Opération non linéaire (S-box) conçu pour résister à la cryptanalyse linéaire et différentielle.
- ***ShiftRow***: Permutation des bytes introduisant des décalages variables sur les lignes.
- ***MixColumn***: Chaque colonne est remplacée par des combinaisons linéaires des autres colonnes (multiplication des matrices!)
- ***AddRoundKey:*** XOR de la matrice courante avec la sous-clé correspondante à l'étape courante.

Détail d'une étape (*round*) AES (II)



Source: Un nouvel algorithme de chiffrement, par Joan Daemen et Vincent Rijmen. Pour la Science,Dossier l'art du secret.

AES: Fonctionnement Global

- Le nombre d'étapes d'AES varie en fonction de la taille de la clé. Pour une clé de 128 bits, il faut effectuer 10 étapes. Chaque augmentation de 32 bits sur la taille de la clé, entraîne une étape supplémentaire (14 étapes pour des clés de 256 bits).
- La decryption consiste en appliquer les opérations inverses dans chacune des étapes (*InvSubBytes*, *InvShiftRows*, *InvMixColumns*). *AddRoundKey* (à cause du XOR) est sa propre inverse.
- Le ***Key Schedule*** consiste en:
  - *Une opération d'expansion de la clé principal*. Si N_e est le nombre d'étapes (dépendant de la clé), une matrice de 4 lignes et 4 * (N_e + 1) colonnes est générée.
  - *Une opération de sélection de la clé d'étape:* La première sous-clé sera constituée des 4 premières colonnes de la matrice générée lors de l'expansion et ainsi de suite.
- *Pseudo-code pour AES*:

```
Rijndael(State,CipherKey)
{
    KeyExpansion(CipherKey,ExpandedKey);            // Key Schedule
    AddRoundKey(State,ExpandedKey[0..3]);           // Premier XOR avec la 1ère sous-clé
    For( i=1 ; i<Ne ; i++ ) Round(State,ExpandedKey[4*i ... (4*i)+3]);// Ne - 1 étapes
    FinalRound(State,ExpandedKey[4*Ne ... 4*Ne+3]). // Dernière étape (pas de MixColumn)
}
```

AES: Remarques Finales et Attaques (I)

- La plus grande force de AES réside dans sa simplicité et dans ses performances, y compris sur des plate-formes à capacité de calcul réduite (p.ex. des cartes à puces avec des processeurs à 8 bits).
- Depuis sa publication officielle, des nombreux travaux de cryptanalyse ont été publiés avec des résultats très intéressants. En particulier, N. Courtois et P.Pieprzyk[1] ont présenté une technique appelée XSL permettant de représenter AES comme un système de 8000 équations quadratiques avec 1600 inconnues binaires. L'effort nécessaire pour casser ce système est estimé (il s'agit encore d'une conjecture...) à 2^{100}.
- Ces attaques se basent sur le caractère fortement algébrique (et largement contesté...) de AES. De plus, il suffit de quelques *known plaintexts* pour les mettre en place, ce qui les distingue des attaques linéaires et différentielles (page 89).
- Ces dernières années (2009-2011) des attaques basées sur des clés similaires (*related key attacks*) [2] ont obtenu des résultats intéressants sur des versions réduites d'AES.
- Une autre famille d'attaques dénommée *side channel attacks* [3]agissant directement sur l'implémentation de l'algorithme permet d'extraire des informations d'intérêt cryptographique lors de l'exécution de l'encryption.

1. Nicolas Courtois and Josef Pieprzyk. Cryptanalysis of Block Ciphers with Overdefined Systems of Equations. Asiacrypt 2002.
2. Alex Biryukov; Orr Dunkelman; Nathan Keller; Dmitry Khovratovich; Adi Shamir (2009-08-19). "Key Recovery Attacks of Practical Complexity on AES Variants With Up To 10 Rounds
3. Dag Arne Osvik, Adi Shamir and Eran Tromer (2005-11-20). Cache Attacks and Countermeasures: the Case of AES

AES: Remarques Finales et Attaques (II)

- Récemment (2015) une attaque de type *Meet in the Middle* (page 90) basé sur des structure bi-cycliques[1] [2] a montré qu'il était possible de réduire l'effort nécessaire pour trouver une clé AES-128 à 2^{126}, soit un facteur 4 par rapport au brute force. Ceci reste tout de même largement au dessus des capacités de calcul actuelles.
- Une autre famille d'attaques dénommée *side channel attacks* [3]agissant directement sur l'implémentation de l'algorithme permet d'extraire des informations d'intérêt cryptographique lors de l'exécution de l'encryption.
- La sécurité de AES (comme pour tout autre algorithme d'encryption) se base toujours sur l'hypothèse d'une clé d'entropie maximale. Les attaques publiées récemment sur des protocoles basés sur AES (comme WPA2) exploitent la faiblesse des passwords/passphrases qui sont à l'origine des clés utilisées par l'algorithme.

---

1. A. Bogdanov et al. "Bicicle Cryptanalysis of the Full AES". Asiacrypt 2011.
2. Biaoshuai Tao et Hongjun Wu, "Improving the Biclique Cryptanalysis of AES". Information Security and Privacy. Volume 9144 of the series Lecture Notes in Computer Science pp 39-56. June 2015.
3. Dag Arne Osvik, Adi Shamir and Eran Tromer (2005-11-20). Cache Attacks and Countermeasures: the Case of AES

Techniques de Cryptanalyse sur les *Block Ciphers*

Cryptanalyse Différentielle[1]

- Il s'agit d'une attaque *chosen plaintext* qui s'intéresse à la propagation des différences dans deux *plaintexts* au fur et à mesure qu'ils évoluent dans les différentes étapes de l'algorithme.
- Il attribue des probabilités aux clés qu'il "devine" en fonction des changements qu'elles induisent sur les ciphertexts. La clé la plus probable a des bonnes chances d'être la clé correcte après un grand nombre de couples *plaintext/ciphertext*.
- Nécessite 2^{47} couples *chosen plaintext* pour obtenir des résultats corrects.

Cryptanalyse Linéaire[2]

- Il s'agit d'une attaque *known plaintext* qui crée un simulateur du bloc à partir des approximations linéaires. En analysant un grand nombre de paires *plaintext/ciphertexts*, les bits de la clé du simulateur ont tendance à coïncider avec ceux du *block cipher* analysés (calcul probabiliste)
- Pour DES une attaque basée sur cette technique nécessite 2^{38} *known plaintexts* pour obtenir une probabilité de 10% de deviner juste et 2^{43} pour un 85%!
- Il s'agit de l'attaque analytique la plus puissante à ce jour sur les *block ciphers*.

---

1. Biham, E. and A. Shamir. (1990). Differential Cryptanalysis of DES-like Cryptosystems. Advances in Cryptology — CRYPTO '90. Springer-Verlag. 2–21.
2. Mitsuru Matsui: Linear Cryptoanalysis Method for DES Cipher. EUROCRYPT 1993: 386-397.

Techniques de Cryptanalyse sur les *Block Ciphers* (II)

- La mise en pratique des attaques différentielles et linéaires présente des difficultés dans la parallélisation des calculs par rapport à une recherche exhaustive de la clé.
- Ces deux attaques sont très sensibles au nombre d'étapes du *block cipher*: les chances de réussite augmentent exponentiellement au fur et à mesure que le nombre d'étapes de l'algorithme diminue.
- Une conjecture très répandue parmi les cryptographes est que ces attaques, à l'époque inédites, étaient connues par les concepteurs des DES. En particulier, le design des S-boxes offre une résistance très grande aux deux techniques.

Attaque Meet-in-the-Middle

- S'applique aux constructions du type $y := E_{K^2} (E_{K^1} (x))$. L'espace de clés pour cette solution est de $\{0,1\}^{112}$. On construit d'abord deux listes L_1 et L_2 de 2^{56} messages de la forme: $L_1 = E_{K^1} (x)$ et $L_2 = D_{K^2} (y)$ avec E et D les opérations d'encryption et decryption respectivement. Il faut alors identifier des éléments qui se répètent dans les deux listes et vérifier notre hypothèse avec un deuxième *known plaintext*. Les K_1 et K_2 associées à cette paire de *known plaintexts* seront (en toute vraisemblance) les clés recherchées!
- Effort nécessaire à réaliser les attaques: 2^{57} opérations pour établir les deux listes + 2^{56} blocs de 64 bits de stockage pour mémoriser les résultats intermédiaires... nettement inférieur au 2^{112} estimé intuitivement...
- Ces techniques *meet-in-the-middle* sont aussi appliquées à la cryptanalyse interne des *block ciphers*.

Public Key Cryptography (Asymmetric Cryptography) Outlook

- Mathematical Background
- Reference Problems
- Factoring
- RSA Encryption Algorithm
- ElGamal Encryption Algorithm
- Rabin Encryption Algorithm
- Elliptic Curbs

Fondements Mathématiques

- **Theorème Fondamental de l'Arithmétique:**

  Tout nombre entier strictement positif n s'écrit de façon unique (à l'ordre près) comme un produit de puissances de nombres premiers p_i distincts, à savoir:

$$n = p_1^{e1} \cdot p_2^{e2} \cdot p_3^{e3} \cdot \ldots \cdot p_m^{em}$$

- **Fonction Phi d'Euler**:

  Soit $n \in Z^+$, la **fonction phi d'Euler $\Phi(n)$** est égale au nombre de d'entiers positifs plus petits que n qui sont relativement premiers à n.

- Calcul de la **valeur** de la fonction **phi d'Euler $\Phi(n)$:**

  D'après le théorème fondamental de l'arthmétique, tout nombre entier n>1 s'écrit:

$$n = \prod_{i=1}^{m} p_i^{ei}$$

  alors $\Phi(n)$ se calcule**:**

$$\Phi(n) = \prod_{i=1}^{m} (p_i^{ei} - p_i^{(ei-1)})$$

Fondements Mathématiques (II)

- En particulier, si **n** s'écrit comme **n = p · q** avec **p** et **q** premiers, alors:

$$\Phi(n) = (p - 1) \cdot (q - 1)$$

- **Théorème d'Euler**:

  Soient $n \in Z^+$ et $a \in Z$ avec pgcd (a, n) = 1, alors on a:

$$a^{\Phi(n)} \equiv 1 \bmod n$$

- **Petit Théorème de Fermat** (cas particulier du théorème d'Euler si n est premier):
  Soient $a \in Z$ et p un nombre premier tel que p ne divise pas a, alors on a:

$$a^{p-1} \equiv 1 \bmod p$$

  A noter que puisque p est premier, on a $\Phi(p) = p-1$.

- **Réduction des exposants mod Φ(n):**
  Si n est le produit de premiers distincts et $r, s \in Z$ t.q. $r \equiv s \bmod \Phi(n)$ alors $\forall a \in Z$:

$$a^r \equiv a^s \bmod n$$

- **Application du Théorème d'Euler au calcul des inverses**:
  Suite au théorème d'Euler, on a que:

$$a\, a^{\Phi(n)-1} \equiv 1 \bmod n$$

  ce qui signifie que $a^{\Phi(n)-1}$ **est l'inverse de a mod n.** En particulier, a^{p-2} **est l'inverse de a mod n si p est premier.**

Fondements Mathématiques (III)

- **Définition:** Le **groupe multiplicatif** de $\mathbf{Z_n}$, noté $\mathbf{Z_n^*}$ est = { **a** $\in \mathbf{Z_n}$ t.q. **pgcd(a,n) = 1}.**
  En particulier, si **n** est **premier:** $\mathbf{Z_n^* = \{ a}$ t.q. $\mathbf{1 \leq a \leq n-1\ }\}$
- Le **nombre d'éléments** ou **ordre** du **groupe multiplicatif** $\mathbf{Z_n^*}$ est Φ**(n)** (par déf. de Φ).
- **Définition**: Soit **a** $\in \mathbf{Z_n}$, l'**ordre** de **a** est le plus petit entier positif **t** pour lequel:

  $$\mathbf{a^t \equiv 1 \bmod n}$$
- **Définition**: Soit $\alpha \in \mathbf{Z_n^*}$, si l'ordre de α est Φ**(n)**, alors α est un **générateur** de $\mathbf{Z_n^*}$.
  Lorsqu'un groupe $\mathbf{Z_n^*}$ a un générateur, on dit qu'il est **cyclique**.
- **Propriétés des générateurs**:
  - $\mathbf{Z_n^*}$ a un générateur ssi. $\mathbf{n = 2, 4, p^k}$ **ou** $\mathbf{2p^k}$, avec **p premier**, $\mathbf{p \neq 2}$ et $\mathbf{k \geq 1}$.
    En particulier, si p est premier, $\mathbf{Z_p^*}$ a un générateur.
  - Si α est un générateur de $\mathbf{Z_n^*}$, alors tous les éléments de $\mathbf{Z_n^*}$ s'écrivent:

    $$\mathbf{Z_n^* = \{\ \alpha^i \bmod p\ t.q\ 0 \leq i \leq \Phi(n) - 1\ \}}$$
  - Le nombre de générateurs de $\mathbf{Z_n^*}$ est $\Phi(\Phi$**(n)).**
  - α est un générateur de $\mathbf{Z_n^*}$ ssi. $\forall$ **premier p** t.q. **p divise** Φ**(n), on a:**

    $$\alpha^{\Phi(n)/p} \neq 1 \bmod n\ .$$

    En particulier si **n** est un **premier** de la forme **2p + 1** avec **p premier** (un tel **n** est appelé un ***safe prime***), α est générateur de $\mathbf{Z_n^*}$ ssi. $\alpha^2 \neq 1$ **mod n** et $\alpha^p \neq 1$ **mod n.**

Calcul des Inverses **mod n**

- ***Fast exponentiation***: En utilisant la représentation binaire d'un nombre, on peut calculer des puissances très efficacement, p.ex calcul de 2^{644} mod 645:

  $(644)_{10} = (1010000100)_2$

  Maintenant, on calcule les exposants correspondants aux puissances de 2, soient:

  $2 \equiv 2 \mod 645$
  $2^2 \equiv 4 \mod 645$
  $2^4 \equiv 16 \mod 645$
  $2^8 \equiv 256 \mod 645$
  $2^{16} \equiv 391 \mod 645$
  ...
  $2^{512} \equiv 256 \mod 645$

  D'après la représentation binaire, on calcule:

  $2^{644} = 2^{512+128+4} = 2^{512}2^{128}2^4 = 1601536 \equiv 1 \mod 645$

  La complexité de cet algorithme *fast exponentiation* est **O ($\log^3 n$).**
  En s'appuyant sur le théorème d'Euler, le calcul de l'inverse d'un nombre dans un tel groupe est donc effectué en temps polynomial.

- **L'algorithme d'Euclide étendu** peut être également utilisé pour trouver un x t.q: **ax** ≡ 1 **mod n** puisque cette congruence s'écrit: **ax - 1 = kn** et donc:

  ax - kn = 1 = pgcd (a, n). La complexité de cet algorithme est également **O ($\log^3 n$).**

Théorème des Restes Chinois

- Le **Théorème des Restes Chinois** (III^{e} siècle!) permet de résoudre des systèmes linéaires de congruences simultanées. Il résout des problèmes soulevés dans des anciens puzzles chinois. Il s'agissait, par exemple, de trouver un nombre qui produit un reste de 1 lorsqu'il est divisé par 3, de 2 lorsqu'il est divisé par 5 et de 3 lorsqu'il est divisé par 7...Il fut également utilisé pour calculer le moment exact d'alignement de plusieurs astres ayant des orbites (et donc des périodes) différentes.
- **Théorème des Restes Chinois**: Soient $n_1, n_2, ..., n_t \in Z^+$ premiers deux à deux (c.à.d. pgcd $(n_i,n_j) = 1$, $i \neq j$) et $a_1, a_2, ..., a_t \in Z$. Alors, le système de congruences:

  $x \equiv a_1 \bmod n_1$

  $x \equiv a_2 \bmod n_2$

  ...

  $x \equiv a_t \bmod n_t$

  a une solution unique x mod N := $n_1\ n_2\ ...\ n_t$

- **Algorithme de Gauss** (1801) pour le calcul de x: $x = \sum_{i=1}^{t} a_i N_i M_i \bmod N$ avec $N_i = N/n_i$ et $M_i = N_i^{-1} \bmod n_i$. La complexité de cet algorithme est **O (log^3 n)**.
- Il est donc possible en temps polynomial de passer des congruences **mod n_i** et aux congruences **mod N**!

Problèmes de Base

- **Problèmes génériques principaux**
  - **Factorisation** (FACTP): Etant donné un entier positif n, trouver sa factorisation en nombre premiers.
  - **Logarithmes discrets** (DLP): Etant donné un nombre premier p, un générateur $\alpha \in Z_p^*$ et un élément $\beta \in Z_p^*$, trouver l'entier x, $0 \leq x \leq p-2$, tel que: $\alpha^x \equiv \beta \bmod p$.
  - **Racine carrée dans Z_n si n est composite** (SQROOTP): Etant donné un entier composite n et un résidu quadratique `a`, trouver la racine carrée de `a mod n`.
- **Problèmes spécifiques** (propres à un système de cryptage):
  - **RSA** (RSAP): Etant donné un entier positif n = pq, un entier positif e avec gcd(e,(p-1)(q-1)) = 1 et un entier c, trouver un entier m avec $m^e \equiv c \bmod n$.
  - **Diffie-Hellman** (DHP): Etant donnée un nombre premier p, un générateur $\alpha \in Z_p^*$ et les éléments $\alpha^a \bmod p$ et $\alpha^b \bmod p$, trouver $\alpha^{ab} \bmod p$.
- **Résultats prouvées**:
  - DHP $\leq_p$ DLP (Equivalent sous certaines conditions[1])
  - RSAP $\leq_p$ FACTP (Prouvé équivalent pour le problème générique, voir page 102)
  - SQROOTP $\cong_p$ FACTP

1. *The Relationship Between Breaking the Diffie--Hellman Protocol and Computing Discrete Logarithms*. U. Maurer, S. Wolf. SIAM Journal of Computing, volume 28, no. 5, pages 1689-1721, 1999.

Classical Factoring Techniques

Exponential Time

They run at best in **O**(exp(c * ln(n))

- Trial Division
- Erastosthenes' Sieve (II B.C.)
- Fermat's Difference of Squares Method (~1650)
- Square Form Factorization (1971)
- Pollard's `p-1` method (1974)
- Pollard's `Rho` Method (1975)

Subexponential Time

They run at best in **O**(exp(c * $(\ln(n))^{1/3}$)

- Continued Fractions (1975)
- Quadratic Sieve (1981)
- Number Field Sieve - NFS (1990)
- General Number Field Sieve - GNFS (2006)

Polynomial Time Methods

- Shor's Algorithm in a Quantum Computer (1994) runs in **O**($\log^c n$)

Factoring: New Developments

- Bernstein's specific NFS purpose computer[1] that (on his assumptions) factors a 1536 bit number would take the same time than a 512-bit computation in a conventional machine running NFS
- Lenstra, & al.[2] claim that the "Berstein factor should be brought down to 1.17 but defend that a 1024-bit number one-day factoring machine can be built for a few thousand dollars under some optimistic assumptions.
- More on the Berstein - Lenstra, Shamir controversy on `http://cr.yp.to/nfscircuit.html`
- Largest number factorization to date (2011) using NFS is RSA-768 (a 232-digit number)[3]. Using hundreds of general purpose processors in a 3 year effort.
- Factoring in a Quantum Computer
  - Significant problems to build the machine (errors, dispersion, etc.)
  - I. Chuang (IBM,Almaden) builds a 7 *qu-bit* computer (December 2001)
  - Works in progress: feasibility of a computer featuring millions of *qu-bits*...?

---

1. Daniel J. Bernstein. *Circuits for Integer Factorization*: a Proposal (October 2001)
2. Lenstra, Shamir et al. *Analysis of Bernstein's Factorization Circuit*. June 2002
3. Kleinjung, et al (2010-02-18). *Factorization of a 768-bit RSA modulus*. International Association for Cryptologic Research. Retrieved 2010-08-09

Procédé d´Encryption/Decryption de RSA[1]

Génération des clés

- Chaque entité (**A**) crée une paire de clés (publique et privée) comme suit:
  - **A** choisit la taille du **modulus n** (p.ex. taille (n) = 1024 ou taille (n) = 2048).
  - **A** génère deux nombres premiers **p** et **q** de grande taille (~ n/2).
  - **A** calcule **n := pq** et **Φ(n) = (p-1)(q-1).**
  - **A** génère l'exposant d'encryption **e**, avec **$1 < e < \Phi(n)$** t.q. **pgcd (e, Φ(n)) = 1**.
  - **A** calcule l'exposant de decryption **d**, t.q.: **$ed \equiv 1 \bmod \Phi(n)$** avec l'algorithme d'Euclide étendu ou avec l'algorithme *fast exponentiation* (page 95).
  - Le couple **(n,e)** est la **clé publique de A**; **d** est la **clé privée de A**.

Encryption

- L'entité **B** obtient **(n,e),** la clé publique authentique de **A**.
- **B** transforme son plaintext en une série d'entiers **m_i**, t.q. **$m_i \in [0, n-1]$** $\forall$i.
- **B** calcule le ciphertext **$c_i := m_i^e \bmod n$**, $\forall$i avec l'algorithme *fast exponentiation*.
- **B** envoie à **A** tous les ciphertext **c_i**.

Decryption

- **A** utilise sa clé privée pour calculer les plaintexts **$m_i = c_i^d \bmod n$**.

---

1. *A Method for Obtaining Digital Signatures and Public Key Cryptosystems*. R.Rivest, A.Shamir and L.M. Adleman. Communications of tha ACM 21 (1978), 120-126

Preuve de RSA

- Soit **m** le plaintext et **c** le ciphertext avec $\mathbf{c := m^e \bmod n}$, il s'agit de prouver:

  $\mathbf{m != c^d \bmod n}$:

  En substituant c par sa valeur on obtient:

  $$\mathbf{c^d \bmod n = m^{ed} \bmod n (*)}$$

  mais, on sait que:

  $$\mathbf{ed \equiv 1 \bmod \Phi(n)}$$

  et donc par définition des congruences, il existe un entier k avec:

  $$\mathbf{ed - 1 = k\Phi(n)}$$

  en substituant dans (*):

  $$\mathbf{(c^d \equiv m^{k\Phi(n)+1} \equiv m^{k\Phi(n)} m) \bmod n}$$

  Si **pgcd (m,n) = 1**, on a par le théorème d'Euler:

  $$\mathbf{1 \equiv m^{\Phi(n)} \bmod n}$$

  donc:

  $$\mathbf{(c^d \equiv m^{k\Phi(n)} m \equiv m) \bmod n}$$ **c.q.f.d.!**

  Si **pgcd (m,n)** $\neq$ **1**, **m** est nécessairement multiple de **p** ou de **q** (cas très peu probable...), on peut montrer en faisant les calculs **mod p** et **mod q** que la congruence reste vraie.

RSA: Sécurité

- Le problème **RSAP** (page 97) consistant à trouver **m** à partir de **c** n'est pas prouvé comme étant aussi difficile que la factorisation mais...:
- ... on peut prouver que si on trouve **d** on peut facilement calculer **p** et **q**. Ceci équivaut à dire que factoriser **n** et trouver **d** nécessitent un effort de calcul équivalent.
- On sait que les méthodes les plus rapides pour factoriser (page 98) ont une complexité *sub-exponentielle* (~ $\mathbf{O(\exp(c * (\ln(n))^{1/3})}$). Le problème reste donc calculatoirement impossible pour des modulus ≥ 1048 bits (2048 bits est un choix fréquent pour uns sécurité durable...).
- Afin d'améliorer la vitesse d'encryption, on a tendance à choisir des exposants **e** assez petits (typiquement: **e := 3, e:=17** et **e:=19**). On a cependant prouvé que le calcul d'une *i-ème racine* (avec **i** petit) modulo un composite **n** peut être nettement plus facile que la factorisation de **n**[1]. Par contre, en 2008 on a prouvé que **la résolution générique du problème RSA est équivalent à la factorisation**.[2]
- L'exposant de decryption **d** doit impérativement être de grande taille[3] (au moins la moitié de la taille de n) pour garantir la sécurité du système.
- Par conséquent, **l'encryption est normalement nettement plus rapide que la decryption** puisque les exposants utilisés sont beaucoup plus petits!

1. *Breaking RSA May Not Be Equivalent to Factoring*. D. Boneh et R. Venkatesan. Advances in Cryptology - EUROCRYPT '98.
2. *Breaking RSA Generically is Equivalent to Factoring.* D. Aggarwal and U. Maurer. CRYPTO 2008 Rump Session.
3. *Cryptanalysis of short RSA secret exponents*. M.J Wiener. IEEE transactions on Information Theory (1990), 553-558.

RSA: Attaques

- Lors qu'on souhaite encrypter le même message pour un groupe de correspondants, il convient d'introduire des variations (*randomization*) avant l'encryption pour éviter l'attaque suivant:
  Admettons qu'on calcule des ciphertexts $\mathbf{c_1,c_2,c_3}$ à partir du même plaintext m et du même exposant **e:=3** adressés à trois entités avec des modulus: $\mathbf{n_1, n_2, n_3}$.
  Le *Théorème des Restes Chinois* nous dit qu'il existe une solution $\mathbf{x \bmod n_1n_2n_3}$, t.q.:
  $\mathbf{x \equiv c_1 \bmod n_1}$, $\mathbf{x \equiv c_2 \bmod n_2}$ $\mathbf{x \equiv c_3 \bmod n_3}$
  Mais si **m** ne change pas pour les trois encryptions, on a que $\mathbf{x = m^3 \bmod n_1n_2n_3}$ et, de plus: $\mathbf{m^3 < n_1n_2,n_3}$. On peut, donc, trouver **m** en calculant la **racine cubique entière** de $\mathbf{m^3}$, en sachant que pour ce calcul il existe des algorithmes efficaces!
- Plus généralement, si $\mathbf{m < n^{1/e}}$, on peut appliquer des algorithmes rapides (dans Z) pour calculer les **racines e-ièmes** de m^e. Il convient donc d'effectuer des opérations de "*randomization*" de **m** avant d'encrypter!
- La propriété multiplicative de RSA: $\mathbf{(m_1m_2)^e \equiv m_1^e\, m_2^e \equiv c_1\, c_2 \bmod n}$
  donne lieu à des failles dangereuses (voir signatures aveugles).
- En admettant que les paramètres sont correctement choisis et que l'implantation n'a pas de failles, la méthode la plus efficace pour "casser" l'algorithme générique RSA reste la factorisation de n.

Procédé d´Encryption/Decryption d'ElGamal[1]

Génération des clés

- Chaque entité (**A**) crée une paire de clés (publique et privée) comme suit:
  - **A** génère un nombre premier **p** de grande taille (**len(p)** $\geq$ 1024 **bits**) et un générateur α du groupe multiplicatif $\mathbf{Z_p^*}$.
  - **A** génère un nombre aléatoire **a**, t.q. $\mathbf{1 \leq a \leq p\text{-}2}$ et calcule $\alpha^{\mathbf{a}}$ **mod p**.
  - La clé publique de **A** est **(p,** α**,** $\alpha^{\mathbf{a}}$ **mod p)**, la clé privée de **A** est **a**.

Encryption

- L'entité **B** obtient **(p,** α**,** $\alpha^{\mathbf{a}}$ **mod p),** la clé publique authentique de **A**.
- **B** transforme son plaintext en une série d'entiers $\mathbf{m_i}$, t.q. $\mathbf{m_i} \in \mathbf{[0, p\text{-}1]}$ $\forall$i.
- Pour chaque message $\mathbf{m_i}$:
  - **B** génère un nombre aléatoire unique **k**, t.q. $\mathbf{1 \leq k \leq p\text{-}2}$.
  - **B** calcule λ **:=** $\alpha^{\mathbf{k}}$ **mod p** et δ **:=** $\mathbf{m_i}$ $(\alpha^{\mathbf{a}})^{\mathbf{k}}$ **mod p** et envoie le ciphertext **c:= (**λ**,**δ**).**

Decryption

- **A** utilise sa clé privée **a** pour calculer $\lambda^{\mathbf{p\text{-}1\text{-}a}}$ **mod p** (à noter que: $\lambda^{\mathbf{p\text{-}1\text{-}a}} \equiv \lambda^{\mathbf{-a}} \equiv \alpha^{\mathbf{-ak}}$ **mod p**).
- **A** retrouve le plaintext en calculant: $(\alpha^{\mathbf{-ak}})$ δ **mod p.**

1. *A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logaithms*. T. ElGamal. Advances in Cryptology-Proceedings of CRYPTO 84 (LNCS 196), 10-18, 1985.

Procédé d'ElGamal: Remarques

- La preuve que la decryption fonctionne est évidente en substituant δ par sa valeur:

$$(\alpha^{-ak})\ \delta \textbf{ mod p} \equiv (\alpha^{-ak})\ \textbf{m}\ (\alpha^{a})^{k} \textbf{ mod p} \equiv \textbf{m mod p.}$$

- Le procédé d'ElGamal se base sur la difficulté de calculer des logarithmes discrets modulo un nombre premier (voir problème **DLP** en page 97) même s'il n'a pas été prouvé qu'il soit strictement équivalent à ce problème.
- Les algorithmes les plus efficaces connus à cette date ont une complexité *sub-exponentielle* très proche de celle de la factorisation (on utilise souvent les mêmes algorithmes...)
- Les exposants choisis **(k,a)** doivent être de grande taille car il existe des algorithmes efficaces pour calculer des logarithmes discrets modulo un nombre premier lorsque l'exposant est petit (*baby-step giant-step algorithm*[1])
- Un inconvénient d'ElGamal est qu'il multiplie par 2 la longueur du ciphertext.
- Il est essentiel pour la sécurité du procédé que le nombre aléatoire **k ne soit pas répété,** autrement: soient (λ_1,δ_1) et (λ_2,δ_2) les deux ciphertext générés, on a que $\delta_1/\delta_2 = \mathbf{m_1/m_2}$ et par conséquent, il est trivial de retrouver un plaintext à partir de l'autre...
- Le procédé d'ElGamal peut se généraliser à d'autres groupes comme **GF(2^n)** ou les courbes elliptiques (voir page 112).

1. Algorithm by D.Shanks appearing in: *The Art of Computer Programming-Fundamental Algorithms*. D.E Knuth. Volume 1. Addison Wesley 1973.

Procédé d´Encryption/Decryption de Rabin[1]

Génération des clés

- Chaque entité (**A)** crée une paire de clés (publique et privée) comme suit:
  - **A** génère deux nombres premiers aléatoires **p** et **q** de grande taille (len (pq) ≥ 1024).
  - **A** calcule **n := pq**.
  - La clé publique de **A** est **n** la clé privée de **A** est **(p,q)**.

Encryption

- L'entité **B** obtient **n,** la clé publique authentique de **A**.
- **B** transforme son plaintext en une série d'entiers $\mathbf{m_i}$, t.q. $\mathbf{m_i} \in \mathbf{[0, n-1]}$ $\forall$i.
- **B** calcule $\mathbf{c_i = m_i^2 \bmod n}$ pour chaque message $\mathbf{m_i}$.
- **B** envoie tous les ciphertext $\mathbf{c_i}$ à **A**.

Decryption

- **A** utilise sa clé privée **(p,q)** pour retrouver les **4 solutions** de l'équation: $\mathbf{c_i = x^2 \bmod n}$ en utilisant des algorithmes efficaces pour calculer des racines carrées **mod p** et **mod q**.
- **A** détermine soit par une indication supplémentaire de **B**, soit par une analyse de redondance **lequel des 4 messages $\mathbf{m_1}$, $\mathbf{m_2}$, $\mathbf{m_3}$, $\mathbf{m_4}$** est le plaintext original.

---

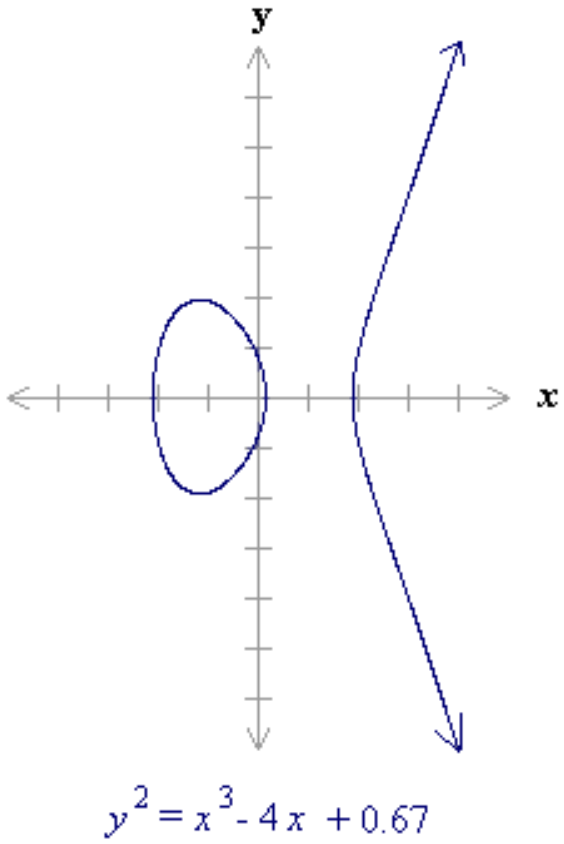
1. *Digitalized Signatures and Public Key Functions as Intractable as Factorization.* M.O.Rabin. MIT/LCS/TR 212. MIT Laboratory for Computer Science 1979.

Procédé de Rabin: Remarques

- Le procédé de Rabin est basé sur l'impossibilité de trouver des racines carrées modulo un composite de factorisation inconnue (problème SQROOTP, voir page 43).
- L'intérêt principal de cet algorithme réside dans le fait qu'il a été prouvé comme étant équivalent à la factorisation (SQROOTP ≅ FACTP). Cet algorithme appartient donc à la catégorie ***provably secure*** **pour toute attaque passive**.
- Les **attaques actives** peuvent, dans certains cas, compromettre la sécurité de l'algorithme. Plus précisément, si on monte l'attaque *chosen ciphertext* (on demande à A de decrypter un ciphertext choisi) suivant:
  - L'attaquant **M** génère un **m** et envoie à **A** le ciphertext $\mathbf{c = m^2 \bmod n}$.
  - **A** répond avec une racine $\mathbf{m_x}$ parmi les 4 possibles $\mathbf{m_1, m_2, m_3, m_4}$.
  - Si $\mathbf{m \equiv \pm m_x \bmod n}$ (probabilité 0.5), **M** recommence avec un nouveau **m**.
  - Sinon, **A** calcule $\mathbf{pgcd\ (m - m_x, n)}$ et obtient ainsi un des deux facteurs de **n**...
- Cette attaque pourrait être évitée si le procédé exigeait une redondance suffisante dans les plaintexts permettant à A d'identifier sans ambiguïté laquelle des solutions possibles est le plaintext original. Dans ce cas, A répondrait toujours **m** et jetterait les autres solutions n'ayant pas le niveau de redondance préétabli.

Courbes Elliptiques[1]

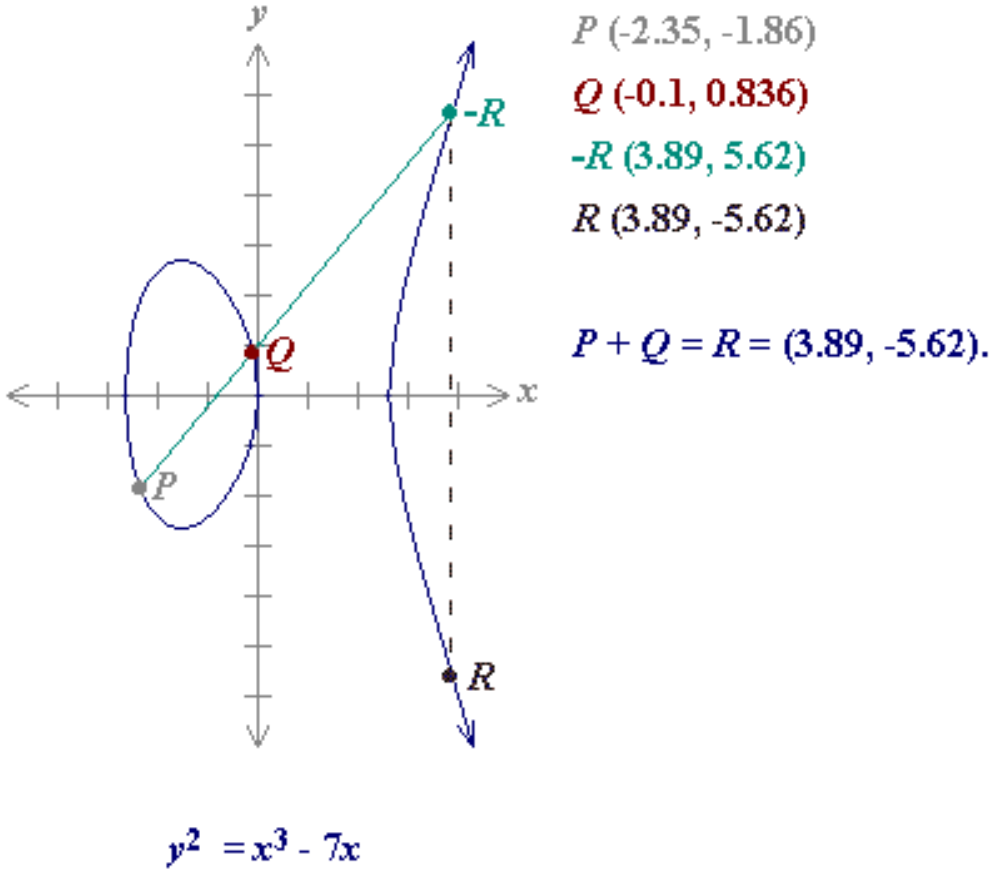
- Une **courbe elliptique** est un ensemble de points **E** défini par l'équation: $\mathbf{y^2 = x^3 + ax + b}$, avec x, y, a et b des nombres **rationnels**, **entiers** ou **entiers modulo m** (m > 1). L'ensemble E contient également un ¨**point à l'infini**¨ noté ∞. Le point ∞ n'est pas dans la courbe mais il est l'élément identité de E.
- On choisira pour nos calculs les courbes elliptiques n'ayant pas de racines multiples ou, en d'autres termes, des courbes où le discriminant $\mathbf{4a^3 + 27b^2 \neq 0.}$



1. Source de toutes les images sur les courbes elliptiques: http://www.certicom.com

Courbes Elliptiques: Addition

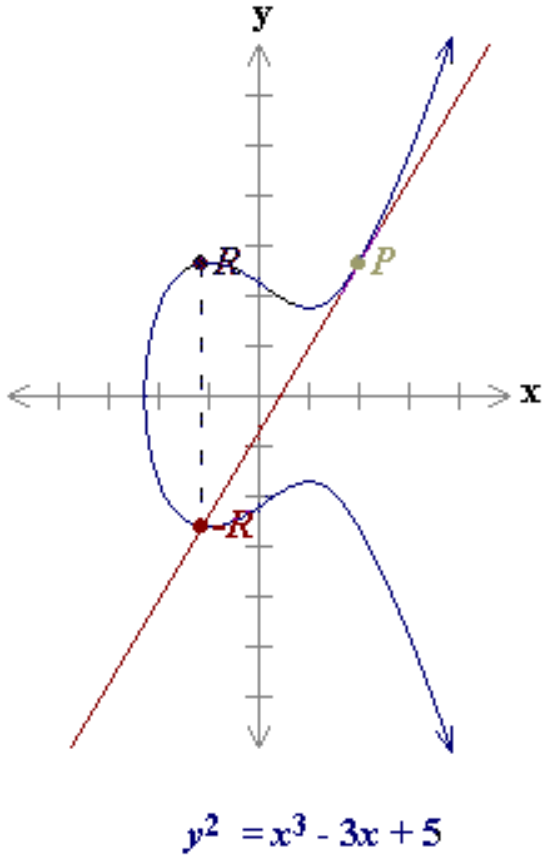
- Soit P := **(x, y)** ∈ **E**, on définit **-P** comme **-P := (x, -y)**. Graphiquement, **-P** est le point symétrique de **P** par rapport à l'axe des **x**. A noter que **P + (-P) = ∞**.
- Soient deux points **P, Q** ∈ **E**, tels que **Q ≠ -P,** on définit l'addition **P + Q := R** où **R** ∈ **E** t.q. **-R** est le 3^{e} point d'intersection entre la courbe et la droite qui passe par **P** et **Q**:



- L'ensemble E avec ∞ définit un **groupe commutatif** pour l'addition.

Courbes Elliptiques: Multiplication par un Scalaire

- Soit P ∈ **E,** le point **2P = R**, t.q. **-R** est le point d'intersection de la courbe avec la droite tangente à la courbe au point P.



P (2, 2.65)

-*R* (-1.11, -2.64)

R (-1.11, 2.64)

$2P = R = (-1.11, 2.64)$.

$y^2 = x^3 - 3x + 5$

- Lorsque la courbe elliptique est définie sur le corps $\mathbf{Z_p}$ avec **p** un nombre premier de grande taille ($\mathbf{y^2 \equiv x^3 + ax + b \bmod p}$), le calcul de $\mathbf{k} \in \mathbf{Z_p}$ t.q. **Q = kP** avec **(P, Q)** connus, est très difficile (nécessite un effort exponentiel...). Ce problème est connu comme: **Elliptic Curve Discrete Logarithm Problem (ECDLP)**

Courbes Elliptiques: Remarques

- L'avantage principal de la cryptographie publique basée sur des courbes elliptiques est que la taille des nombres utilisés (et donc, des clés) est plus petite.
- Ceci est dû à la complexité accrue des calculs sur $\mathbf{E_p}$ (courbe elliptique définie sur le corps $\mathbf{Z_p}$) par rapport aux corps habituels tels que $\mathbf{Z_p}$ ou **GF** $(\mathbf{2^m})$.
- Ce tableau montre les rapports des tailles des clés par rapport à celles de RSA:

NIST guidelines for public key sizes for AES

ECC KEY SIZE (Bits)	RSA KEY SIZE (Bits)	KEY SIZE RATIO	AES KEY SIZE (Bits)
163	1024	1 : 6	
256	3072	1 : 12	128
384	7680	1 : 20	192
512	15 360	1 : 30	256

Supplied by NIST to ANSI X9F1

- La représentation d'un plaintext en points de la courbe reste une opération complexe.
- En Octobre 2003[1], la *US National Security Agency* (NSA) a acheté un brevet de *Certicom* pour l'utilisation de la cryptographie à courbes elliptiques.
- En Septembre 2013 Claus Diem montré[2] que sous certaines condtitions le problème **ECDLP** pouvait être resolu en temps *sub-exponentiel*.

1. http://www.certicom.com/index.php?action=company,press_archive&view=175

2. http://www.math.uni-leipzig.de/~diem/preprints/dlp-ell-curves-II.pdf

Procédé d'ElGamal sur des Courbes Elliptiques

Génération des clés

- Chaque entité (**A**) crée une paire de clés (publique et privée) comme suit:
  - **A** choisit une courbe elliptique $\mathbf{E_p}$ avec **p,** un nombre premier de grande taille (**len(p)** ~ 200 **bits**) et un point $\mathbf{P_o} \in \mathbf{E_p}$.
  - **A** génère un nombre aléatoire **a**, t.q. $\mathbf{1 < a < p}$ et calcule $\mathbf{P_a = aP_o}$ (multiplication par un scalaire sur $\mathbf{E_p}$, pour laquelle, il existe des algorithmes efficaces).
  - La clé publique de **A** est $\mathbf{(E_p, P_o, P_a)}$, la clé privée de **A** est **a**.

Encryption

- L'entité **B** obtient $\mathbf{(E_p, P_o, P_a)}$, la clé publique authentique de **A**.
- **B** transforme son plaintext en une série d'entiers $\mathbf{m_i}$, t.q. $\mathbf{m_i} \in \mathbf{E_p}$ $\forall$i.
- Pour chaque message $\mathbf{m_i}$:
  - **B** génère un nombre aléatoire unique **k**, t.q. $\mathbf{1 < k < p}$.
  - **B** calcule λ **:=** $\mathbf{kP_o}$ et δ **:=** $\mathbf{kP_a + m_i}$ et envoie le ciphertext **c:=** $\mathbf{(\lambda,\delta)}$.

Decryption

- **A** utilise sa clé privée **a** pour calculer: $\mathbf{a\lambda = akP_o = kP_a}$.
- **A** retrouve le plaintext en calculant: δ **-** $\mathbf{kP_a = kP_a + m_i - kP_a = m_i}$.
- Le sécurité du schéma s'appuie sur **ECDLP**!

Tableau Récapitulatif[1]

Table 1

Public-key system	Mathematical Problem	Examples
Integer factorization	Given a number n, find its prime factors	RSA, Rabin-Williams
Discrete logarithm	Given a prime n, and numbers g and h, find x such that $h = g^x \bmod n$	ElGamal, Diffie-Hellman, DSA
Elliptic curve discrete logarithm	Given an elliptic curve E and points P and Q on E, find x such that Q=xP	EC-Diffie-Hellman, ECDSA

Table 2

Public-key system	Best known methods for solving mathematical problem	Running times
Integer factorization	Number field sieve: exp(1.923 (log n)1/3(log log n)2/3)	Sub-exponential
Discrete logarithm	Number field sieve: exp(1.923 (log n)1/3(log log n)2/3)	Sub-exponential
Elliptic curve discrete logarithm	Pollard-rho algorithm: square root of n	Fully exponential

1. *Next Generation Security for Wireless.* S.A Vanstone, 2004. http://www.compseconline.com/hottopics/hottopic20_8/Next.pdf

Hashes and MACs
Outlook

- *One-way* functions
- Hash Function: Definition and classification
- Unkeyed hash functions (MDCs)
  - MDCs based on cryptographic algorithms
  - Customized MDCs (MD5, SHAx)
- Keyed hash functions (MACs)
- Applications
- *Randomized hash functions*: UNIX example

One-way Functions

- Une fonction f est dite à sens unique (***one-way function ou OWF***) si $\forall x \in X$ on peut facilement calculer $f(x) = y$ mais pour la grande majorité des $y \in Y$ il est calculatoirement impossible de trouver un x tq. $f(x) = y$.
- Exemples:
  - calcul des carrées modulo un composite:

    $f(x) = x^2 \bmod n$ avec $n = pq$ (p et q inconnus)

    est une one-way function car l'inverse est difficile (voir le problème de base SQROOTP).
  - on peut construire une one-way function sur la base de DES ou de n'importe quel autre système de cryptage à blocs E comme suit:

    $y = f(x) = E_k(x) \oplus x$, avec k une clé fixée et connue

    on peut considérer que $E_k(x) \oplus x$ a un comportement (pseudo) aléatoire par construction de E. Le calcul de l'inverse revient à trouver un x t.q. :

    $x = E_k^{-1}(x \oplus y)$

    ce qui est considéré difficile avec les propriétés de E.

    A noter que $f(x) = E_k(x)$ ne suffirait pas pour en faire une OWF car, en connaissant la clé, DES est réversible.

Hash Functions: Définitions

- Une fonction de hachage (***hash function***) est une fonction h ayant les propriétés suivantes:
  - **compression**: la fonction h fait correspondre à un ensemble `X` composée par des chaînes de bits **de longueur finie mais arbitraire**, un ensemble Y composé par des chaînes de bits **de longueur finie et fixée** (et normalement inférieur à la taille de `X`) avec `h(x) = y`, et `x ∈ X`, `y ∈ Y`.
  - **facile à calculer**: partant de `h` et `x ∈ X`, `h(x)` est facile à calculer.
- Une hash function est dite "à clé" (***keyed hash function***) si une clé intervient dans le calcul de la fct. ($h_k(x) = y$); sinon on l'appelle "sans clé" (***unkeyed hash function***).
- Les hash functions ont des nombreuses applications informatiques dont l'archivage structuré facilitant la recherche. Coté sécurité nous allons étudier deux catégories principales:
  - codes détecteurs d'altérations (***manipulation detection codes (MDC)*** or *message integrity codes (MIC)*): ce sont des *unkeyed functions* permettant de fournir un service d'intégrité sous certaines conditions. Le résultat d'une telle fonction est appelée *MDC-value* ou, simplement, *digest*.
  - codes d'authentification de message (***message authentication codes ou MAC***) qui sont des *keyed functions* permettant d'authentifier la source du message et d'assurer son intégrité sans utiliser des mécanismes (cryptage) additionnels.

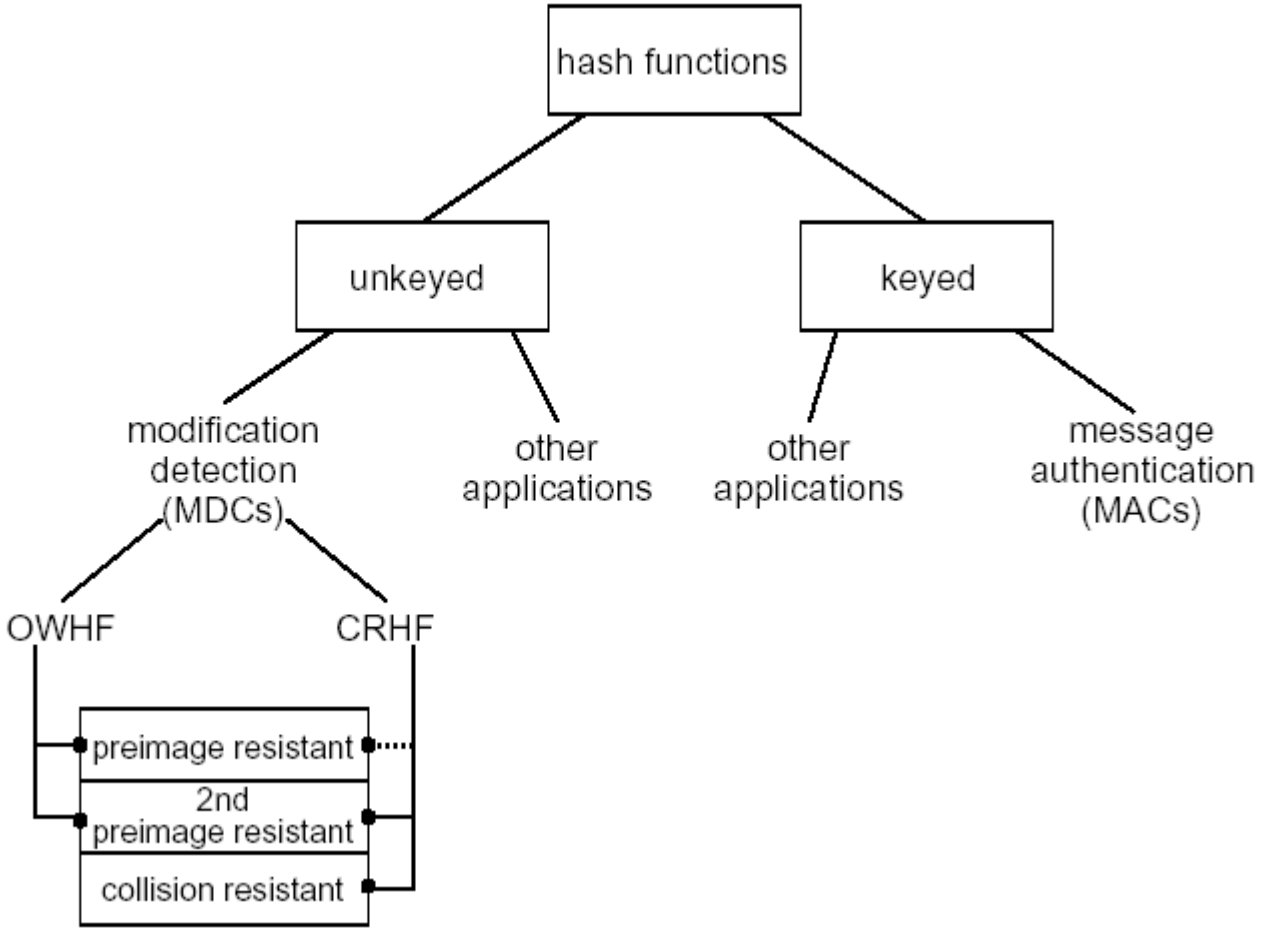
Hash Functions: Définitions (II)

- Quelques propriétés de base des hash functions:
  - 1) *preimage resistance*: étant donné un $y \in Y$, il est calculatoirement impossible de trouver une pré-image $x \in X$ satisfaisant $h(x) = y$.
  - 2) *2nd-preimage resistance*: étant donné un $x \in X$ et son image $y \in Y$, avec $h(x) = y$, il est calculatoirement impossible de trouver un $x' \neq x$ tel que $h(x) = h(x')$. Aussi appelée *weak collision resistance*.
  - 3) *collision résistance*: il est calculatoirement impossible de trouver deux pré-images $x, x' \in X$ distinctes pour lesquels $h(x) = h(x')$ (pas de restriction sur le choix des valeurs). Aussi appelée *strong collision resistance*.
- Une fonction de hachage à sens unique (***one way hash function ou OWHF***) est un MDC satisfaisant l) et 2). Aussi appelée: *weak one-way hash function.*
- Une fonction de hachage résistante aux collisions (***collision resistant hash function ou CRHF***) est un MDC satisfaisant le propriétés 2) et 3). (A noter que 3) $\Rightarrow$ 2)). Aussi appelée: *strong one-way hash function.*
- OWF $\neq>$ OWHF: A noter qu'une OWHF en tant que hash function impose des restrictions supplémentaires sur les domaines sources et image ainsi que sur la 2nd-*preimage resistance* qui ne sont pas forcement respectés par des OWFs.
  - Exemple: $f(x) = x^2 \bmod n$ avec $n = pq$ (p et q inconnus) n'est pas une OWHF car étant donné x, $-x$ est une collision triviale.

Message Authentication Codes (MAC)

- Un **Message Authentication Code** (**MAC**) est une famille de fonctions h_k paramétrisées par une clé secrète k ayant les propriétés suivantes:
  - 1) **compression**: comme pour les fonctions de hash génériques mais appliqué à h_k.
  - 2) **facile à calculer**: à partir d'une fonction h_k, et d'une clé connue k, on peut facilement calculer $h_k(x)$. Le résultat est appelée un *MAC-value* ou, simplement, un *MAC*.
  - 3) **résistance calculatoire** (*computation-resistance*): sans connaissance de la clé symétrique k, il est (calculatoirement) impossible de calculer des paires $(x, h_k(x))$ à partir de 0 ou plusieurs paires connus $(x_i, h_k(x_i))$ pour tout $x \neq x_i$.
- La propriété 3) implique que les paires $(x_i, h_k(x_i))$ ne peuvent non plus servir à calculer la clé k (*key non-recovery*). Cependant la propriété *key non-recovery* n'implique pas *computation-resistance* car des attaques *chosen/known -plaintext* pourraient mener à des paires $(x, h_k(x))$ falsifiées.
- L'impossibilité de calculer des paires $(x, h_k(x))$ se traduit également en *preimage* et *collision resistance* (cf. transparent précédent) pour toute entité ne possédant pas la clé k.

Hash Functions: Schéma Récapitulatif



(Source [Men97])

Attaques sur des MDCs: *2nd preimage résistance*

- Problème: étant donné `h(x) = y`, trouver `x'` tq. `h(x')=h(x)`.
- Exemple pratique: on a un texte avec un *digest* associé portant une signature digitale; on veut créer un faux texte portant la même signature (sans avoir le contrôle sur le texte original). Quelles sont nos chances d'un point de vue probabiliste?
- Soit une hash function `h` avec `n` sorties possibles et une valeur donnée `h(x)`. Si `h` est appliquée à `k` valeurs aléatoires, quelle doit être la valeur de `k` pour que la probabilité d'avoir au moins un `y` tq. `h(x) = h(y)` soit `0.5`?
- Pour la première valeur de `y`, la probabilité que `h(x) = h(y)` est `1/n`. Inversement, la probabilité que `h(x)` ≠ `h(y)` est `1-1/n`. Pour `k` valeurs, la probabilité de n'avoir aucune collision est de: $(1-1/n)^k$, soit:

$$\left(1-\frac{1}{n}\right)^k = 1-k\frac{1}{n}+\binom{k}{2}\frac{1}{n^2}-\binom{k}{3}\frac{1}{n^3}+\ldots$$

  ce qui pour `n` très grand peut être approché par `1 - k/n`. Par conséquent, la probabilité complémentaire d'avoir au moins une collision est d'environ `k/n`; c'est qui nous donne `k = n/2` pour une probabilité de `0.5`.
- Conclusion: pour un digest de `m`-bits, le nombre d'essais nécessaires à trouver un `y` tq. `h(x) = h(y)` avec une probabilité de `0.5` est 2^{m-1}.

Attaques sur des MDCs: *collision résistance*

- Problème: trouver deux valeurs `x`, `x'` distincts t.q. `h(x) = h(x')`.
- Exemple pratique: On doit faire signer un texte à quelqu'un et on veut appliquer cette signature à un texte falsifié (on contrôle le texte original). Quelles sont nos chances de trouver deux textes originaux satisfaisant ce critère?

Dear Anthony,

{This letter is / I am writing} to introduce {you to / to you} {Mr. / —} Alfred {P. / —}
Barton, the {new / newly appointed} {chief / senior} jewellery buyer for {our / the}
Northern {European / Europe} {area / division}. He {will take / has taken} over {the / —}
responsibility for {all / the whole of} our interests in {watches and jewellery / jewellery and watches}
in the {area / region}. Please {afford / give} him {every / all the} help he {may need / needs}
to {seek out / find} the most {modern / up to date} lines for the {top / high} end of the
market. He is {empowered / authorized} to receive on our behalf {samples / specimens} of the
{latest / newest} {watch and jewellery / jewellery and watch} products, {up / subject} to a {limit / maximum}
of ten thousand dollars. He will {carry / hold} a signed copy of this {letter / document}
as proof of identity. An order with his signature, which is {appended / attached}
{authorizes / allows} you to charge the cost to this company at the {above / head office}
address. We {fully / —} expect that our {level / volume} of orders will increase in
the {following / next} year and {trust / hope} that the new appointment will {be / prove}
{advantageous / an advantage} to both our companies.

Exemple d'une lettre falsifiée en 2^{37} variations (source [Sta95])

Birthday Paradox: Fondement Mathématique

- Le *birthday paradox* est un problème probabiliste classique qui montre que dans une réunion de 23 personnes seulement, on a déjà une chance sur deux d'avoir deux personnes ayant leur anniversaire le même jour.
- Soit $y_1, y_2, \ldots, y_n$ toutes les sorties possibles d'une hash function. Combien des $h(x_i)$: $h(x_1), h(x_2), \ldots, h(x_k)$ devons nous calculer pour avoir une probabilité de collision égale ou supérieure à 0.5 ?
- Le premier choix pour $h(x_1)$ est arbitraire ($prob = 1$), le deuxième $h(x_2) \neq h(x_1)$ a une probabilité de $1 - 1/n$, le troisième de $1 - 2/n$, etc. Ce qui nous donne une probabilité de ne pas avoir des collisions égale à:

$$P_{no-collision} = \prod_{i=1}^{k-1}\left(1 - \frac{i}{n}\right)$$

On prouve facilement (développement en série de e^{-x}) que pour $0 \leq x \leq 1: 1-x \leq e^{-x}$ et donc:

$$P_{no-coll} = \prod_{i=1}^{k-1}\left(1 - \frac{i}{n}\right) \leq \prod_{i=1}^{k-1} e^{-\frac{i}{n}} = e^{\frac{-k(k-1)}{2n}}$$

Birthday Paradox: Fondement Mathématique (II)

La probabilité d'avoir au moins une collision est $P_{au\text{-}moins1} = 1 - P_{no\text{-}coll}$, soit:

$$P_{au-moins1} \geq 1 - e^{\frac{-k(k-1)}{2n}}$$

Pour connaître la valeur de k pour laquelle $P_{au\text{-}moins1}$ est plus grand que 0.5, il suffit de calculer:

$$\frac{1}{2} = 1 - e^{\frac{-k(k-1)}{2n}}$$

Si k est grand, on remplace $k(k-1)$ par k^2 et on obtient après des calculs simples:

$$k = \sqrt{2(\ln 2)n} \cong 1,17\sqrt{n} \approx \sqrt{n}$$

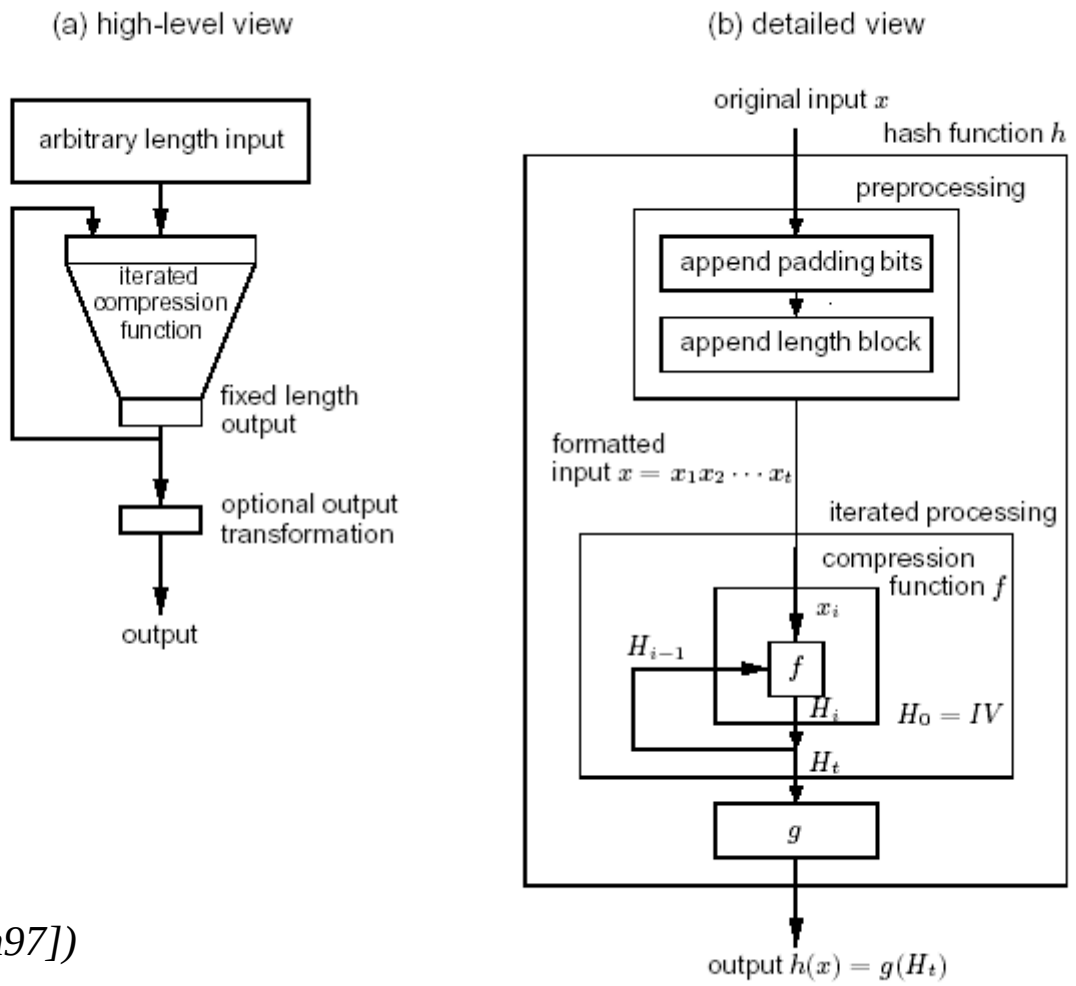
- En prenant n = 365 pour l'anniversaire, on obtient k = 22.3, ce qui confirme l'énoncé du problème
- Conséquence pour les hash functions: Soit une hash function avec 2^m sorties possibles. Si h est appliqué à $k = 2^{m/2}$ entrées on a une probabilité supérieur à 0.5 d'obtenir $h(x_i) = h(x_j)$

Résistance Calculatoire Théorique des Hash Functions: Récapitulation

Type de Hash Fct.	Caractéristique	Difficulté Calculatoire	But de l'attaque	Taille conseillée du digest/clé
OWHF	*preimage resistance*	2^n	trouver une pré-image	$n \geq 80$ bits
	2nd-preimage résistance	2^{n-1}	trouver x' avec $h(x') = h(x)$	
CRHF	*collision resistance*	$2^{n/2}$	trouver une collision	$n \geq 160$ bits
MAC	*key non-recovery*	2^t	trouver la clé	$n \geq 80$
	computation resistance	$\min(2^t, 2^n)$	produire un $(x, h_k(x))$	$t \geq 80$

- n: taille du *MDC-value* ou du *MAC-value* résultant de l'application de la *hash function*
- t: taille de la clé du MAC

Modèle de fonctionnement générique des MDCs

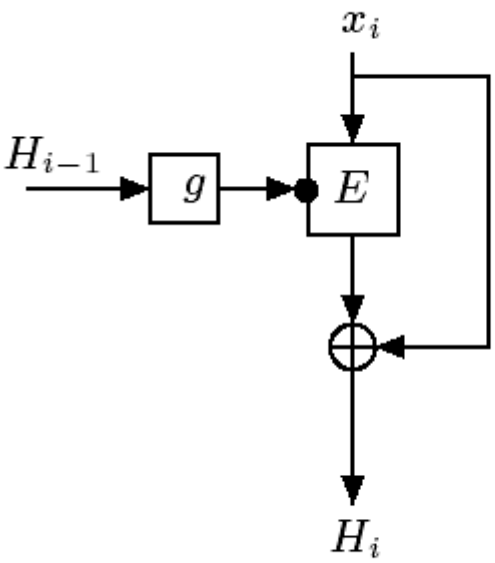


(Source [Men97])

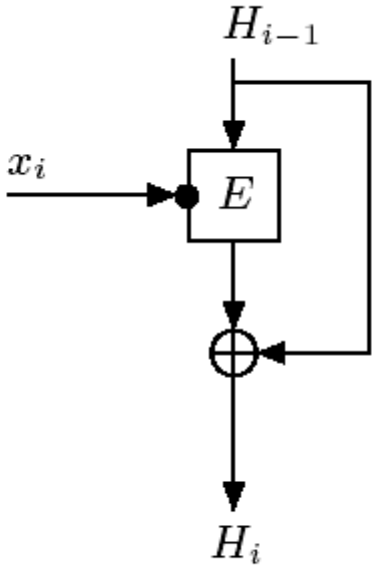
MDCs Basées sur des Systèmes de Cryptage

- Idée: utiliser un système de cryptage symétrique connu pour construire un MDC.
- Problèmes à résoudre:
  - il faut “casser” la réversibilité des algorithmes symétriques pour en faire des OWHF ou des CRHF.
  - La “largeur nominale” de certains systèmes de cryptage (eg. DES) est de 64 bits, ce qui n’est pas suffisant pour construire des CRHF.
- Principe de fonctionnement:
  - les blocs de texte sont séquentiellement traités par la “boîte” de cryptage.
  - la compression se base sur des opérations de chaînage avec les blocs résultant des itérations précédentes et des fonctions logiques (fondamentalement `XOR`). Ceci rend également le procédé irréversible.
  - Si nécessaire, `n` boîtes de cryptage seront combinées pour obtenir des longueurs de *digests* `n` fois supérieures à la largeur nominale des boîtes utilisées.
- Attention: la sécurité de ces algorithmes est fortement dépendante des propriétés des boîtes de cryptage sous-jacents.
- Exemples:
  - Les modèles de *Matyas-Meyer-Oseas*, *Davies-Meyer* et *Miyaguchi-Preneel*.
  - MDC-2 et MDC-4 utilisant respectivement 2 et 4 boîtes DES. *Digest* = 128 bits.

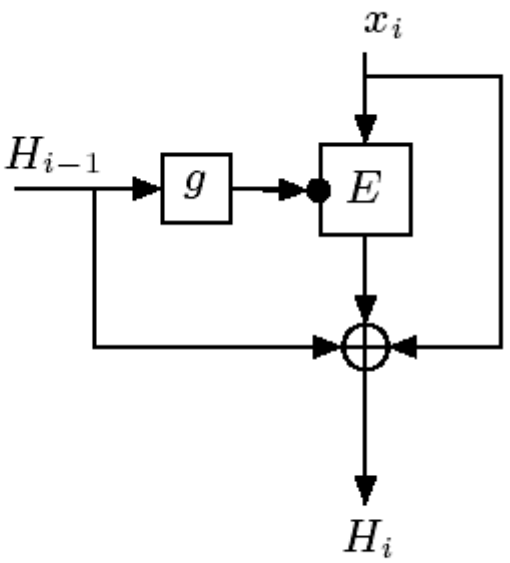
MDCs Basées sur des Systèmes de Cryptage: Modèles Génériques



Matyas-Meyer-Oseas



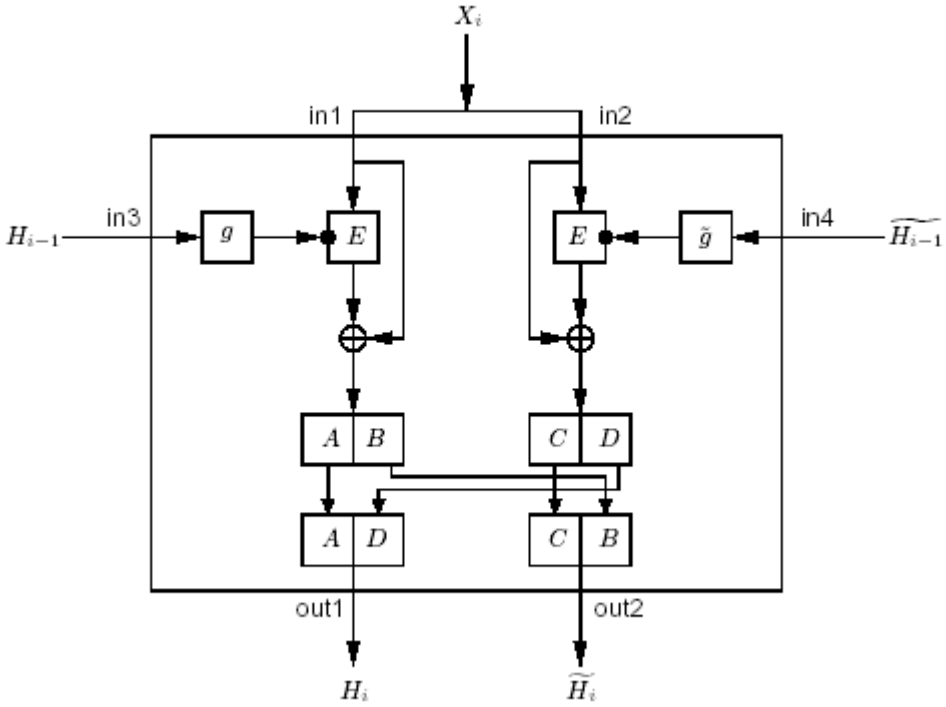
Davies-Meyer



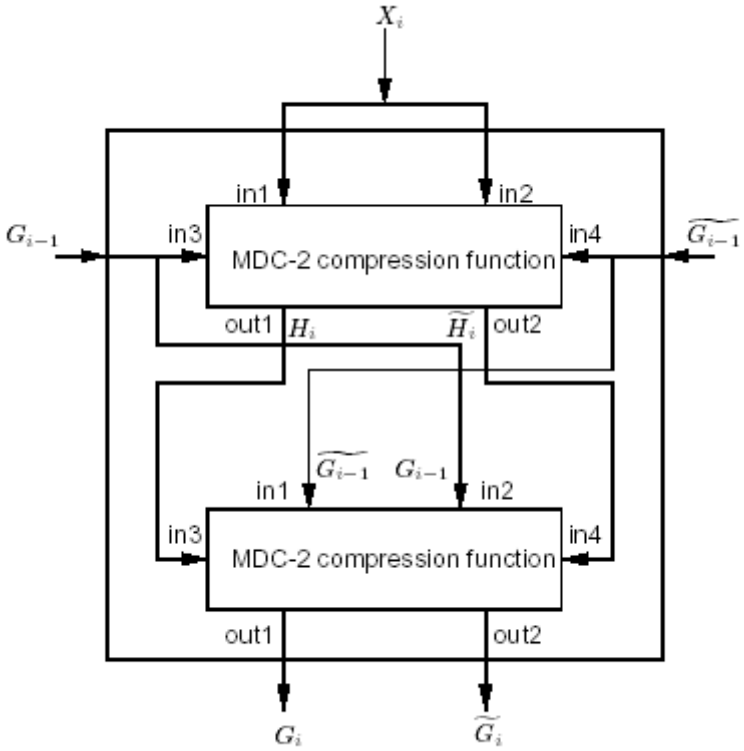
Miyaguchi-Preneel

(source [Men97])

Exemples(II): MDC-2 et MDC-4



Hash Function MDC-2
E-box = DES



Hash Function MDC-4

(Source [Men97])

Customized MDCs

- Il s'agit de fonctions conçues exclusivement pour générer des codes d'intégrité (des *digests*) avec un soucis principal de vitesse et sécurité.
- Leur fonctionnement se base sur les éléments suivants:
  - des opérations d'initialisation (*padding* + rajouter la longueur).
  - un ensemble de constantes prédéfinies choisies spécialement pour augmenter la dispersion.
  - un ensemble "d'étapes" (*rounds*) qui vont séquentiellement s'appliquer a tous les blocs des données originaux. Ces rounds vont effectuer une combinaison d'opérations logiques et des rotations sur les données et les constantes.
  - des opérations de chaînage impliquant les sorties des *rounds* précédents.
- Dans ces fonctions, chaque bit du *digest* est une fonction de chaque bit des entrées.
- Les plus connues sont:
  - MD5: R. Rivest, 1992; RFC 1321. *Digest = 128 bits*. *Cassé!*
  - SHA-0: NIST, 1993. *Digest = 160 bits*. Collisions en 2^{39} opérations au lieu de 2^{80}
  - SHA-1: NIST, 1995. *Digest = 160 bits*. Révision de SHA-0 avec rotation de bits additionnelle. Collisions en 2^{63} opérations (au lieu de 2^{80}).
  - SHA-2: NIST (FIPS 190-3). Comprend: SHA-224, SHA-256, SHA-384 et SHA-512. Les tailles du digest vont de 224 à 512 bits.
  - SHA-3: *Keccak* Algorithm (taille du digest variable de 224 à 512 bits)

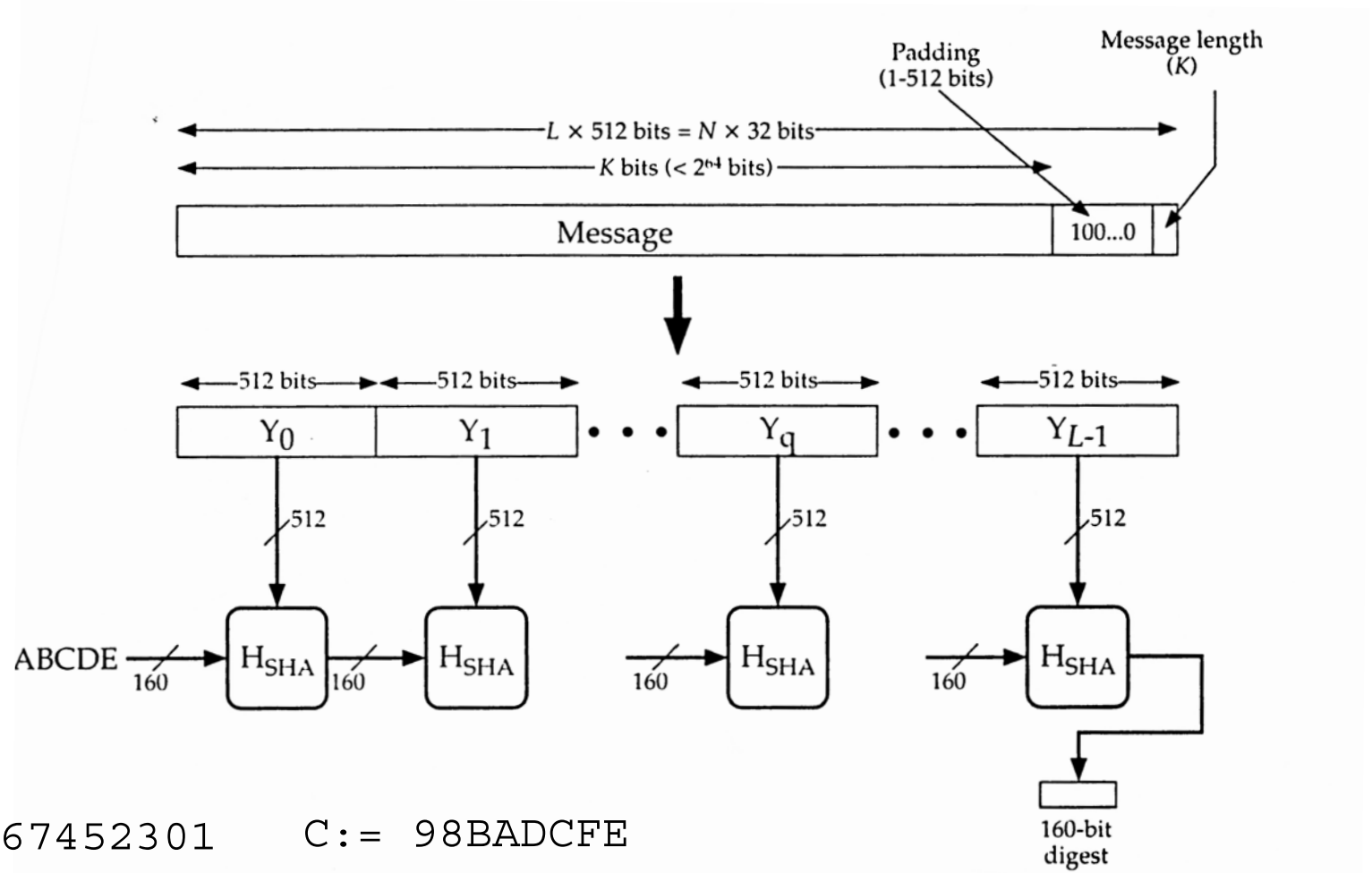
Hash Functions: Derniers Développements

- X.Wang et al.[1] culminent en 2004 un long travail visant à trouver des collisions dans l'algorithme MD5. Ils publient deux paires de collisions pour des messages de 1024 bits.
- En 2005, X.Wang et al.[2] prouvent dans la conférence CRYPTO'05 que le nombre d'opérations nécessaires pour trouver des collisions sur SHA-1 (standard actuel pour les fonctions de hashage sécurisées) est seulement de 2^{63}.
- Ces attaques ont pour cible la recherche de collisions arbitraires mais lors de CRYPTO'06 des chercheurs de l'Université de Graz en Autriche[3] proposent une méthode pour contrôler partiellement le contenu des collisions.
- En Décembre 2008[4] on montre qu'on peut générer des collisions contrôlées sur MD5 et créer ainsi une Certification Authority illicite permettant des forger des certificats acceptés par n'importe quel browser.
- Ces résultats s'appuient sur des approches analytiques (par opposition au *brute force*!)
- Le processus de sélection de successeur de SHA-1 est semblable à celui ayant désigné AES comme standard de cryptage en blocs. Le NIST a décidé (Octobre 2012) que Keccak serait l'algorithme de base pour **SHA-3**.

---

1. X.Wang et al. *Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD*. http://eprint.iacr.org/2004/199.pdf.
2. X.Wang et al. *Finding Collisions in the Full SHA-1*. Advances in Cryptology. Crypto'05
3. C. Cannière et al. *SHA-1 collisions:Partial meaningful at no extra cost?* Ramp Sessions CRYPTO'06.
4. M. Stevens et al.*Short Chosen-Prefix Collisions for MD5 and the Creation of Rogue CA Certificate*. CRYPTO'09

Secure Hash Algorithm: Vue d'Ensemble



A:= 67452301 C:= 98BADCFE

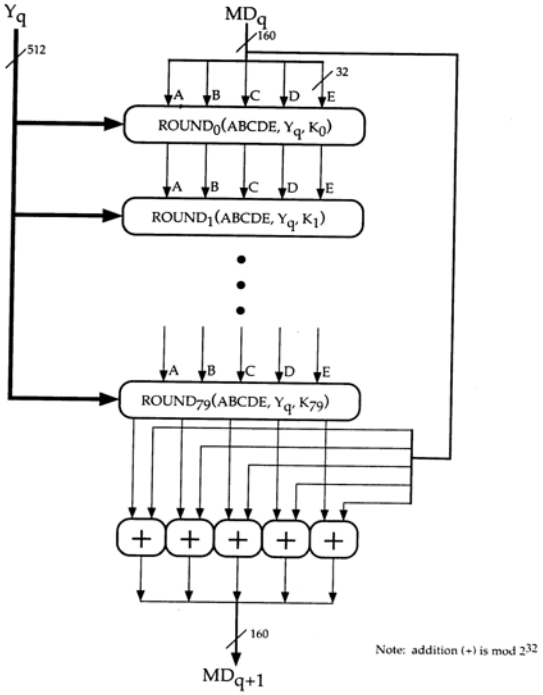
B:= EFCDAB89 D:= 10325476

E:= C3D2E1F0

(Source [Sta95])

SHA-0: Détails de Fonctionnement

SHA: Traitement d'un bloc de 512 bits H_{SHA}



SHA: Opération élémentaire

(Source [Sta95])

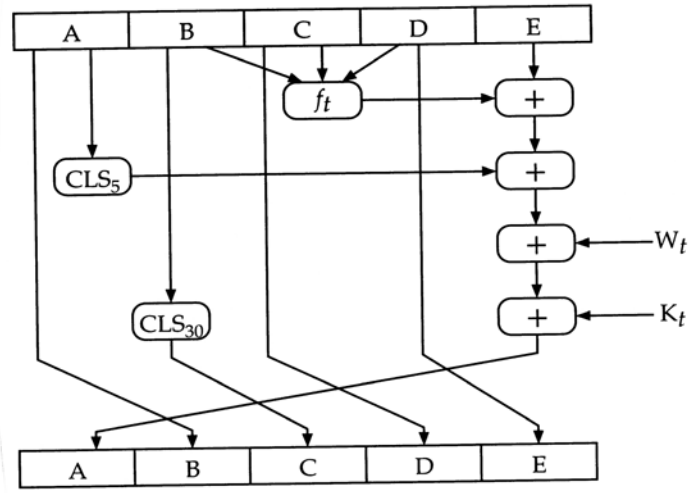


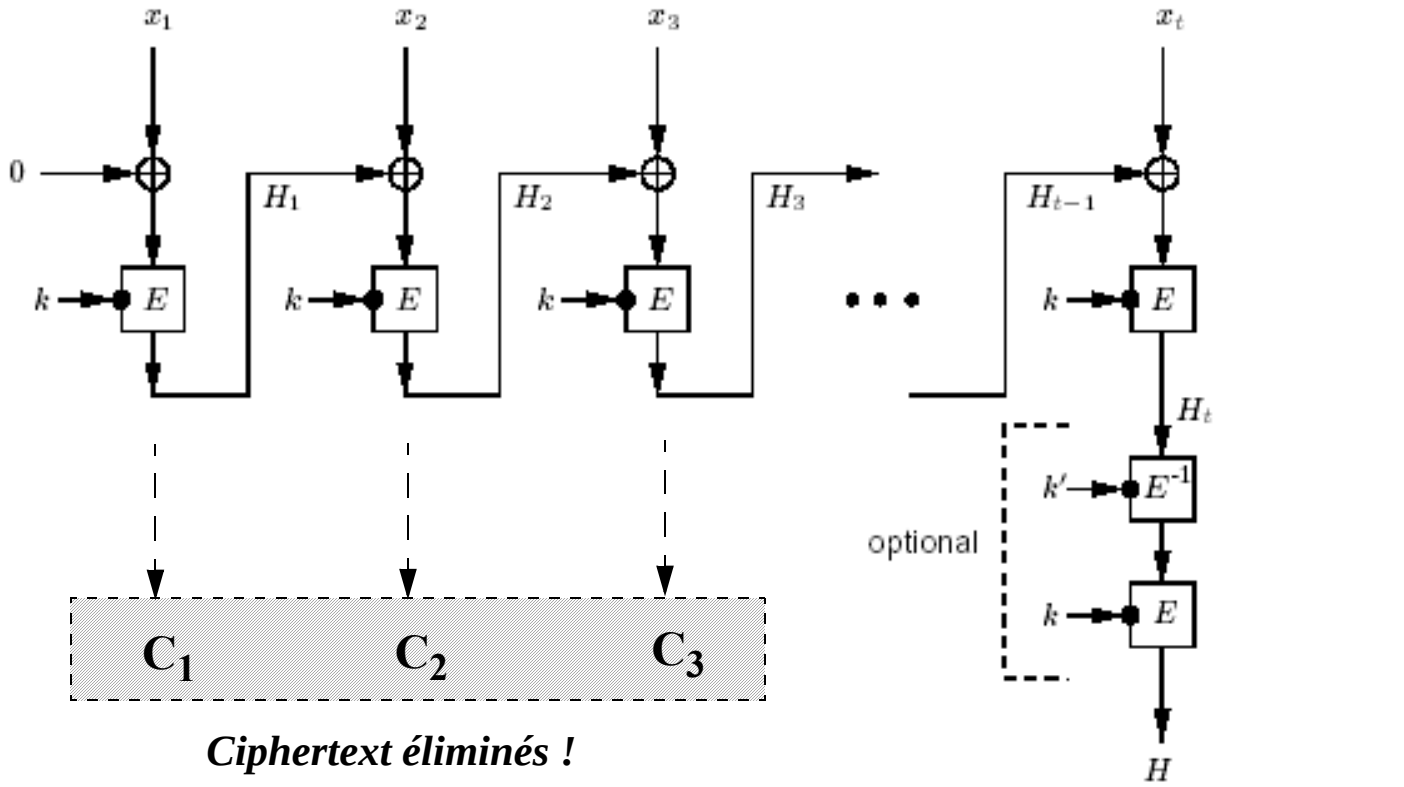
FIGURE 7.7. Elementary SHA Operation

$0 \le t \le 19$	K_t:=5A827999	$f_t(b,c,d)$:= (b·c) U ($\bar{b}$·d)
$20 \le t \le 39$	K_t:=6ED9EBA1	$f_t(b,c,d)$:= b ⊕ c ⊕ d
$40 \le t \le 59$	K_t:=8F1BBCDC	$f_t(b,c,d)$:= (b·c) U (b·d) U (c·d)
$60 \le t \le 79$	K_t:=CA62C1D6	$f_t(b,c,d)$:= b ⊕ c ⊕ d

Les blocs de text W_t se calculent : $\mathbf{W_t := W_{t-16} \oplus W_{t-14} \oplus W_{t-8} \oplus W_{t-3}}$

MACs basés sur des Systèmes de Cryptage

- Algorithme CBC-MAC basé sur DES-CBC avec `IV = 0` et élimination des *ciphertext* intermédiaires
- longueur de clé = 56 bits (112 en cas d'utilisation de la partie optionnelle)
- Longueur du `MAC-value` = 64 bits



Ciphertext éliminés !

(Source [Men97])

Nested MACs et HMACs

- Un ***Nested MAC*** ou ***NMAC*** est une composition de 2 familles de fonctions MACs G et H paramètrès par les clés k et l tel que:

  $G \circ H = \{ g \circ h \text{ avec } g \in G \text{ et } h \in H \}$ avec $g \circ h_{(k,l)}(x) = g_k(h_l(x))$

- La sécurité d'un NMAC dépend de deux critères:
  - La famille de fonctions G est résistante aux collisions.
  - La famille de fonctions H est résistante aux attaques spécifiques pour MACs, i.e.:

    Il est impossible de trouver un couple (x,y) et une clé m fixée mais inconnue, telle que: $MAC_m(x) = y$.

- Un ***HMAC*** (FIPS 198, 2002) est un Nested MAC utilisant à la base des MDCs sans clé dédiées comme SHA-1 ou SHA-256.
- Un HMAC utilise deux constantes de 512 bits dénommés *ipad* et *opad* telles que:

  opad := 363636 ... 36 et ipad := 5C5C5C ... 5C

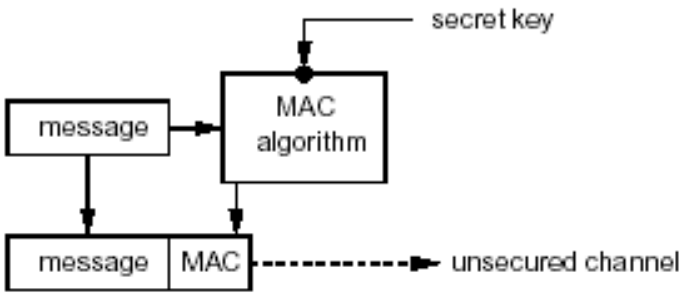
  et une clé k de 512 bits.
- Le schéma de fonctionnement de HMAC-256 (sur la base de SHA-256) est le suivant:

  $HMAC\text{-}256_k(x) := SHA\text{-}256((k \oplus opad) \,||\, SHA\text{-}256((k \oplus ipad) \,||\, x))$

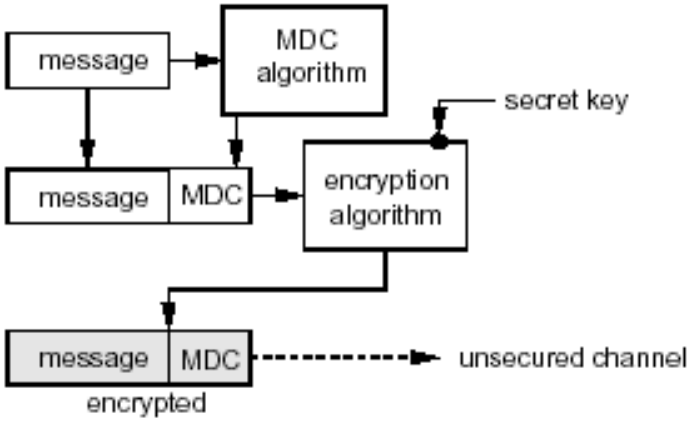
- Les HMACs sont les MACs les plus utilisés. Les attaques mentionnées sur les fonctions de la famille SHA sont plus difficiles à réaliser sur un HMAC par cause de la clé k.

Applications des Hash Functions: Intégrité

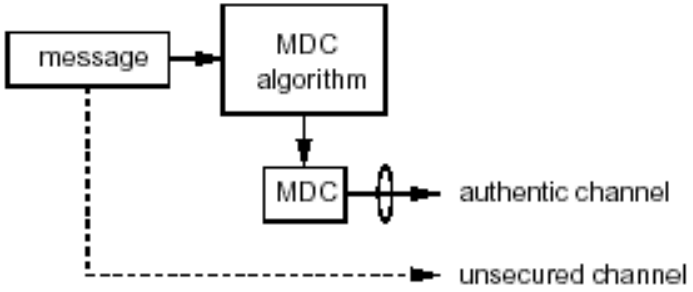
MAC Seul:



MDC + Encryption:



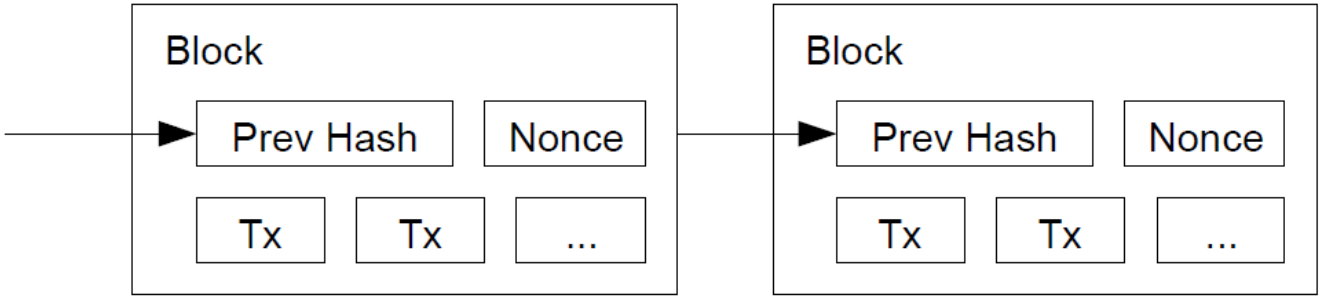
MDC + canal authentique:



(Source [Men97])

Application des *Hash Functions*: *Blockchains*

- Les transactions bitcoin sont publiées et visibles par tous les intervenants. Elles sont encapsulées dans des **blocs** chaînés à l'aide de fonctions de hachage cryptographiques
- Le **minage** (***mining***) consiste à rajouter itérativement des nouveaux blocs contenant les transactions courantes
- La génération d'un bloc valable nécessite la **résolution d'un *puzzle cryptographique (proof of work)*** très coûteux en temps de calcul (trouver des *pseudo-collisions* dans les fonctions de hachage cryptographiques). La validation reste très efficace
- Le premier mineur capable de générer un bloc valable recevra une récompense monétaire (en bitcoins). Le processus de minage est ouvert à tous les mineurs mais seul le premier est récompensé
- La chaine de blocs résultante (**blockchain**) devient alors un **registre publique** (*public ledger*), décentralisé et immuable protégeant toutes les transactions passées. La falsification/modification des données protégées par la *blockchain* nécessiterait un effort calculatoire supérieur à celui effectué par tous les mineurs *honnêtes*

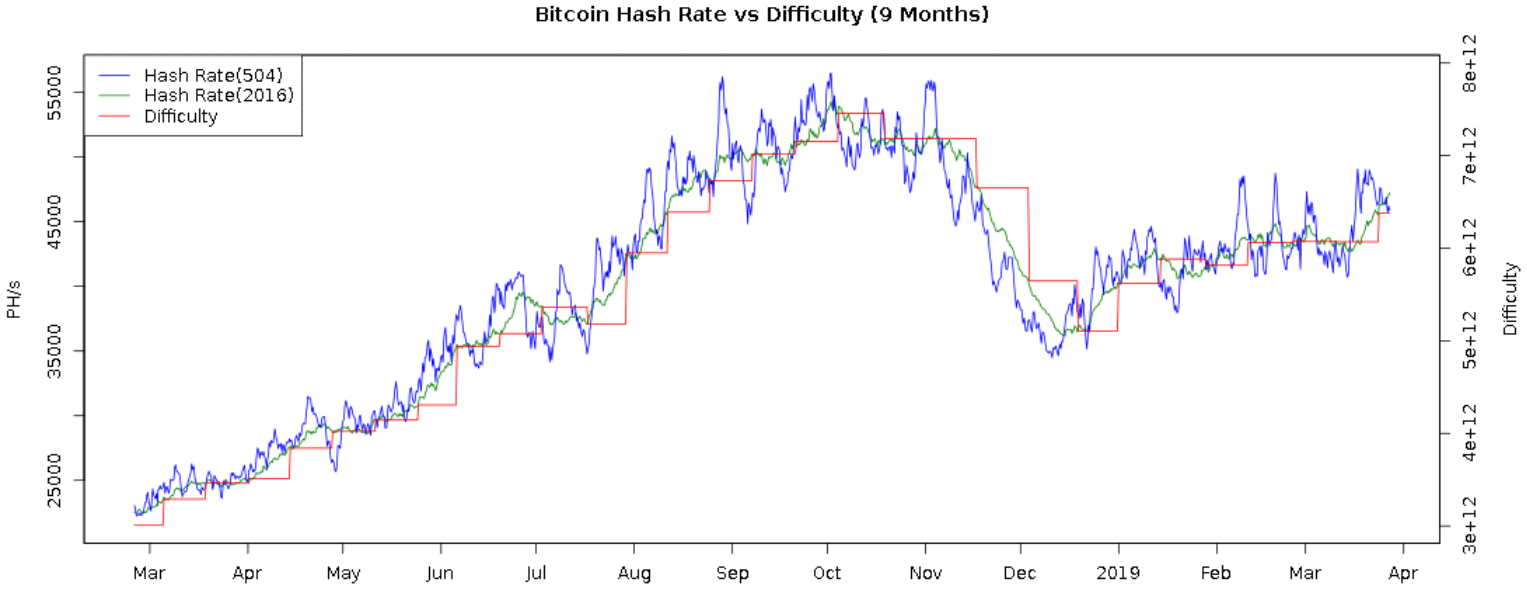


Source Image: *Bitcoin: A Peer-to-Peer Electronic Cash System.* Satoshi Nakamoto

Blockchain: *Proof of Work*

Statistiques Bitcoin 22/10/2019 *(source http://bitcoinwisdom.com):*

- ***Difficulty***: **6,379,265,451,411**
- ***Target***: $\mathbf{2^{224}}$ **/ *Difficulty*** = ~ $\mathbf{2^{181}}$. Le digest valable pour générer un bloc doit être **inférieur à** $\mathbf{2^{181}}$, ce qui signifie une ***pseudo-collision sur les 74 bits de poids plus fort***. La variation sur les inputs dépend du *nonce*
- ***Hashrate***: ~ **46 * $\mathbf{10^{18}}$ hashes /sec**
- Fonctions de hachage exécutées pour obtenir un bloc: ~ **27* $\mathbf{10^{21}}$**
- Temps de génération d'un bloc: **9,9 minutes**



Autres Applications de Hash Functions

- Authentification:
  - *data origin authentication* (DOA)
  - *transaction authentication* (= DOA + *time-variant parameters*)
- *Virus checking*
  - Le créateur d'un logiciel crée un digest = `h(x)` avec `x` étant l'original et le distribue par un canal sûr (eg. CD-ROM).
- Distribution des clés publiques
  - Permet de contrôler l'authenticité d'une clé publique.
- *Timestamp* sur un document:
  - Le document sur lequel on veut effectuer le timestamp est d'abord soumis à une hash function. Le timestamp (avec la signature de l'entité correspondante) s'applique alors seulement au digest.
- *One-time password (S-Key)* (mécanisme d'identification)
  - A partir d'un *seed* secret x_0, on crée une chaîne de hash-values: $x_1 = h(x_0)$, $x_2 = h(x_1)$, ... $x_n = h(x_{n-1})$.
  - Le système mémorise x_n et l'utilsateur rentre x_{n-1}. Si $h(x_{n-1}) == x_n$ => OK.
  - Le système mémorise alors x_{n-1} et ainsi de suite.

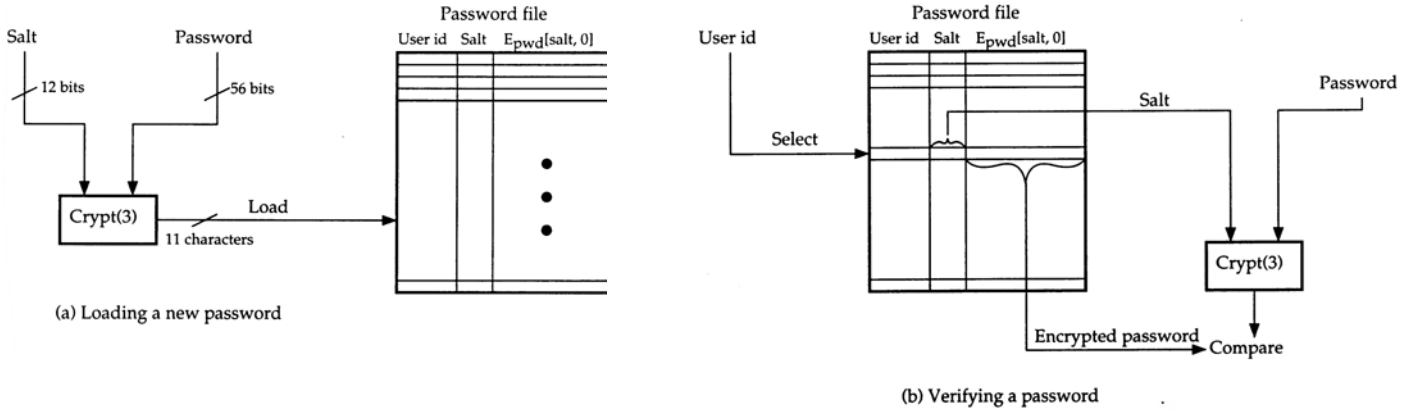
Randomized Hash Functions (l'exemple UNIX) I

- UNIX garde ses mots de passe dans un fichier globalement accessible (ou éventuellement distribué par NIS)
- L'information stockée correspond au résultat produit par une hash function.
- Exemple (fictif):

```
root:Jw87u9bebeb9i:0:1:Operator:/:/bin/csh
pp:1Qhw.oihEtHK6:359:355:PP:/net/spp_telecom/pp:/bin/cs
```

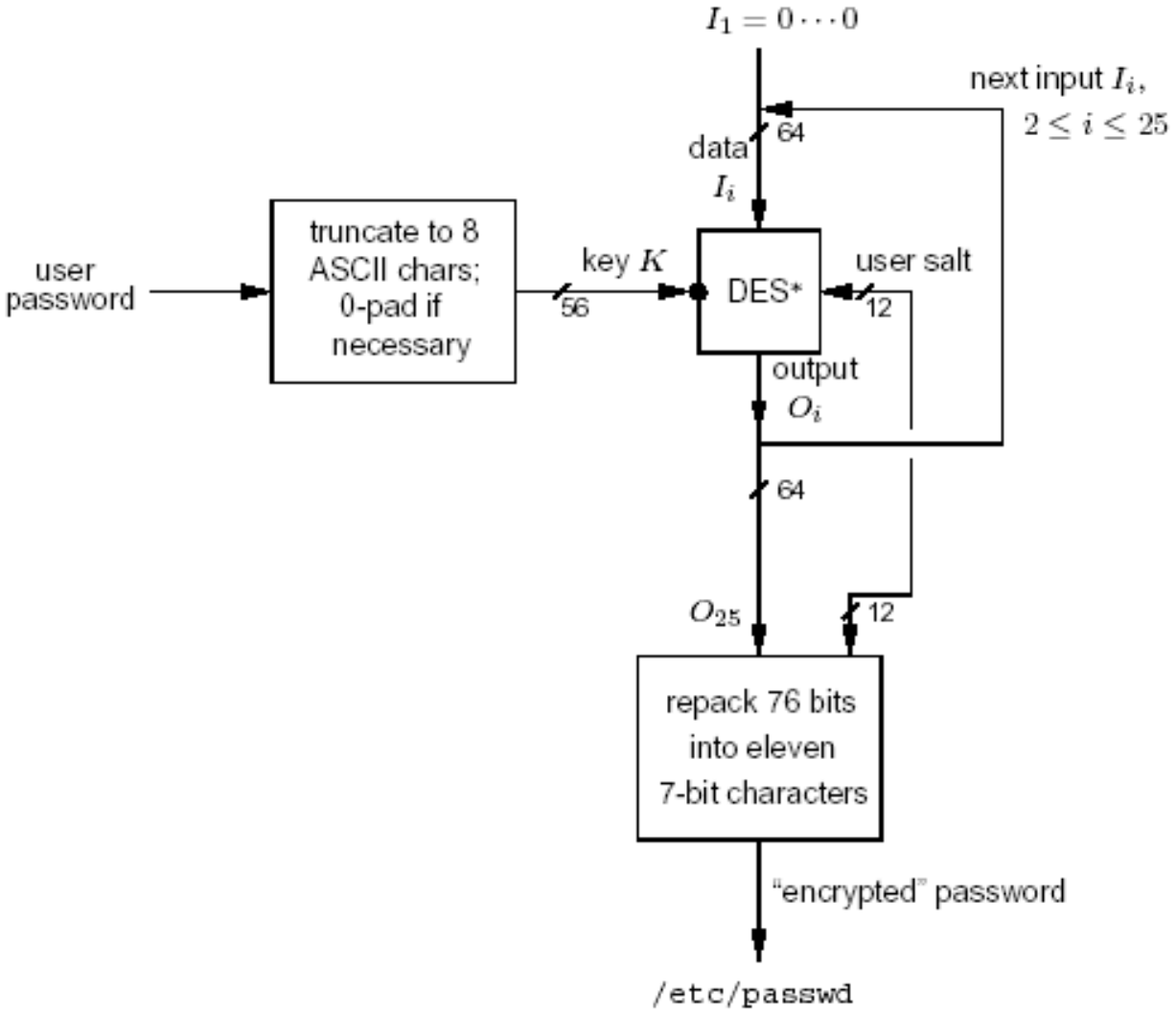
- Problèmes:
  - la hash function étant déterministe, elle produit le même résultat pour des mots de passe identiques.
  - on pourrait créer des "cahiers" (*codebooks*) contenant le résultat de l'application de la hash function à des entrées données (p.ex. un dictionnaire) et les comparer facilement (*off-line*) avec les chaînes stockées par UNIX (*brute force dictionnary attack*).
- Solution:
  - Rajouter un élément (pseudo) aléatoire de 12 bits différent pour chaque mot de passe (appelé *salt*) avant de calculer la hash function et lors de la vérification.
  - Cet élément permet de rajouter un facteur aléatoire de 4096 possibilités pour chaque mot de passe et de prévenir la détection des duplications.

Randomized Hash Functions (l'exemple UNIX) II



(Source [Sta95])

Randomized Hash Functions (l'exemple UNIX) III



(Source [Men97])

Digital Signatures
Outlook

- Definitions
- Digital Signatures with Appendix
- Digital Signatures with Message Recovery
- RSA Signature Algorithm
- Rabin Signature Algorithm
- ElGamal Signature Algorithm
- Digital Signatures and Cryptocurrencies
- Attacks on Digital Signatures

Signatures Digitales: Définitions

- *Signature digitale*: chaîne de données permettant d'associer un message (sous forme digitale) à une entité d'origine.
- *Schéma de signature digitale*: algorithme de génération + algorithme de vérification.
- *Procédé de signature*: formatage du message + algorithme de génération de signature
- *Procédé de vérification*: algorithme de vérification + (reconstruction du message).
- Classification des signatures digitales:
  - *Signatures digitales avec appendice* qui nécessitent la présence du message original pour vérifier la validité de la signature. Ce sont les plus couramment utilisées. Exemples: *ElGamal*, *DSS*.
  - *Signatures digitales avec reconstitution du message* qui offrent, en plus, la possibilité de reconstruire le message à partir de la signature. Exemples: *RSA*, *Rabin.*
- Les signatures digitales sont pour la plupart basées sur la crypto asymétrique du fait que la notion clé partagée n'est pas adaptée aux besoins d'identifier une entité de façon explicite.
- Des engagements semblables à ceux obtenus par une signature à clé publique (comme la non-répudiation d'origine) peuvent cependant être obtenus avec la technologie symétrique et des tierces de confiance (*Trusted Third Parties* ou *TTP*). Ces méthodes sont nommées: *arbitrated digital signatures*.

Signatures Digitales avec Appendice: Cadre Formel

- On admet que chaque entité a une clé privée pour signer des messages et une copie authentique des clés publiques des correspondants.

  Notation: M: Espace de messages
  M_h: $m_h = H(m)$ avec $m \in M$, $m_h \in M_h$ et H une *hash function*
  S: Espace des valeurs pouvant être obtenues par un procédé de signature

  Description:

  Chaque entité définit une appl. injective $S_A : M_h \rightarrow S$; (ie. la *signature*)

  L'application S_A donne lieu à une application V_A:

  $V_A: M_h \times S \rightarrow \{vrai, faux\}$; (ie. la *vérification*)

  t.q. $\forall\ m_h \in M_h, s \in S$, on a:

  $V_A(m_h, s) = vrai$ si $S_A(m_h) = s$ et

  $V_A(m_h, s) = faux$ sinon

  Les opérations S_A nécessitent la clé *privée* de A alors que les opérations V_A utilisent la clé *publique* de A.

- Quelques propriétés simples:

  - Les opérations S_A et V_A doivent être faciles à calculer (en ayant les clés corresp.)

  - Il est impossible (calculatoirement) pour une entité n'ayant pas la clé privée de A de trouver un m' et un s' avec $m' \in M$ et $s' \in S$ t.q. $V_A(m'_h, s') = vrai$ avec $m'_h = H(m')$.

SD avec reconstitution du message: Cadre Formel

Notation: en plus des définitions précédentes, on a:

M_s: L'espace des éléments sur lesquels peut s'appliquer une signature.

R: Une application injective: $M \rightarrow M_s$, appelée *fonction de redondance*. Elle doit être *inversible* et *publique.*

M_R: $M_R = Im(R)$

Description:

Chaque entité définit une appl. injective $S_A : M_S \rightarrow S$; (ie. la *signature*)

L'application S_A donne lieu à une application V_A: ;(ie. la *vérification*)

$$V_A : S \rightarrow M_s \text{ , t.q. } V_A \text{ o } S_A = \text{Identité sur } M_s$$

A noter que la vérification s'effectue sans la clé privée de A

Génération de signature:

(1) Calculer $m_R = R(m)$ et $s = S_A(m_R)$

(2) Rendre publique s en tant que signature de A sur m. Ceci permet aux autres entités de vérifier la signature et reconstituer m.

Vérification:

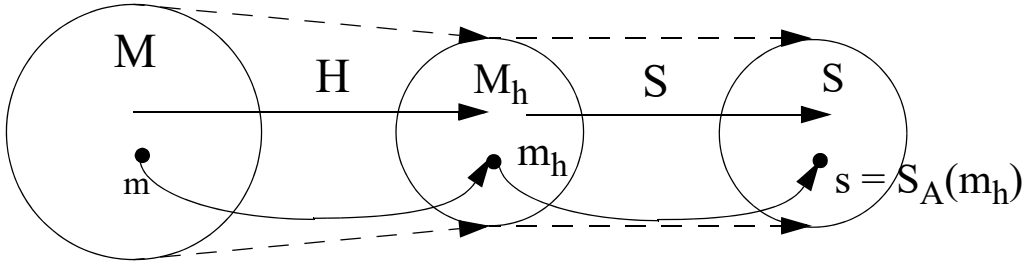
(1) Calculer $m_R = V_A(s)$ (avec la clé publique de A)

(2) Vérifier que $m_R \in M_R$ (sinon rejeter la signature)

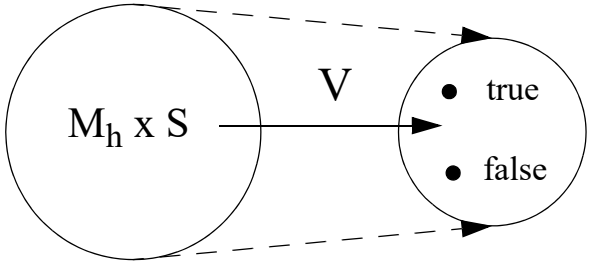
(3) Reconstituer m en calculant: $R^{-1}(m_R)$

Signatures Digitales: Appendice et Reconstitution

Signature Digitale avec appendice

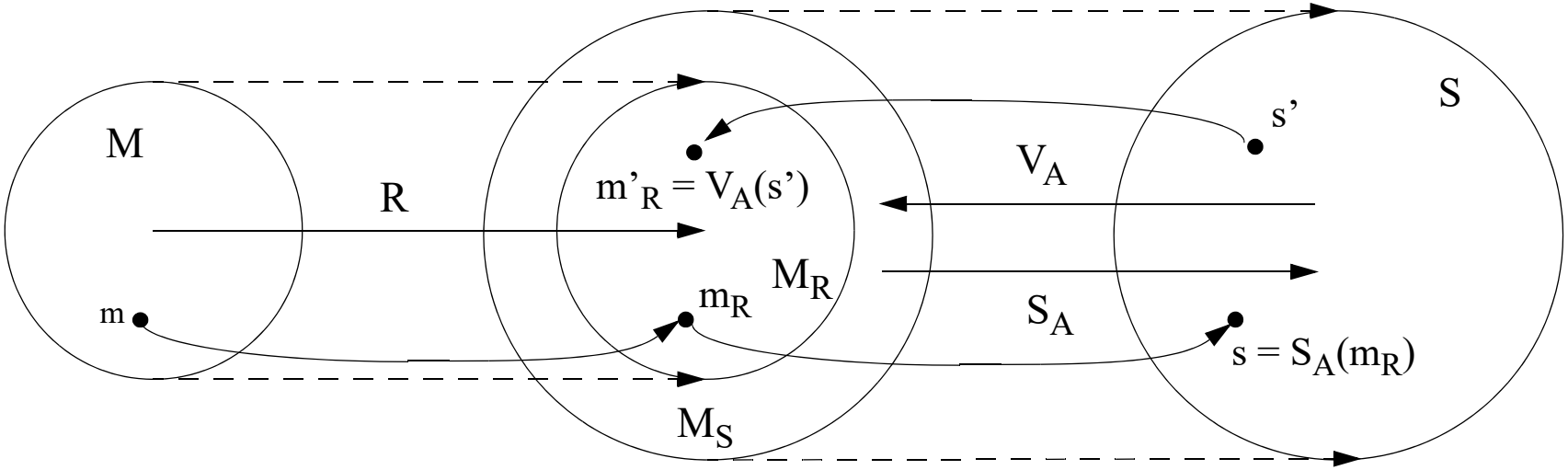


Génération



Vérification

Signature Digitale avec réconstitution du message



SD avec reconstitution du message: Propriétés

- Propriétés:
  - Les opérations S_A et V_A doivent être faciles à calculer (en ayant les clés corresp.)
  - Il est impossible (calculatoirement) pour une entité n'ayant pas la clé privée de A de trouver un s' $\in$ S t.q. $V_A(s') \in M_R$
- Remarques sur la fonction de redondance:
  - Le choix d'une fonction de redondance est essentiel pour la sécurité du système.
  - Si $M_R = M_S$ et R et S_A sont des bijections respectivement de M dans M_R et de M_S dans S, alors M et S ont une taille identique et, par conséquent, il est trivial de forger des messages portant la signature de A.
- Exemple de fonction de redondance: soit $M = \{m: m \in \{0,1\}^n\}$ (n taille du message) et $M_S = \{t: t \in \{0,1\}^{2n}\}$. Soit $R : M \rightarrow M_S$ t.q. $R(m) = m \,||\, m$ (|| étant la concaténation de 2 messages). La probabilité de tomber sur un tel message en essayant de forger un message à partir d'une signature est de : $|M_R| / |M_S| = (1/2)^n$, ce qui est négligeable pour des grands messages.
- Attention!: Une fonction de redondance adaptée pour un schéma de signature digitale peut provoquer des failles dans un autre différent !

Procédé de Signature de RSA[1]

Génération des clés

- Chaque entité (**A**) crée une paire de clés (publique et privée) comme suit:
  - **A** choisit la taille du **modulus n** (p.ex. taille (n) = 1024 ou taille (n) = 2048).
  - **A** génère deux nombres premiers **p** et **q** de grande taille (~ n/2).
  - **A** calcule **n := pq** et **Φ(n) = (p-1)(q-1).**
  - **A** génère l'exposant de vérification **e**, avec **1 < e < Φ(n)** t.q. **pgcd (e, Φ(n)) = 1**.
  - **A** calcule l'exposant de signature **d**, t.q.: **ed $\equiv$ 1 mod Φ(n)** avec l'algorithme d'Euclide étendu ou avec l'algorithme *fast exponentiation* (page 95).
  - Le couple **(n,e)** est la **clé de publique de A**; **d** est la **clé privée de A**.

Signature

- **A** calcule la fonction de redondance du message **m**: $\mathbf{m_R := R(m)}$.
- **A** calcule la signature: $\mathbf{s := m_R^d \bmod n}$ et envoie **s** à **B**.

Vérification

- L'entité **B** obtient **(n,e),** la clé publique authentique de **A**.
- B calcule $\mathbf{m'_R = s^e \bmod n}$, vérifie $\mathbf{m'_R \in M_R}$ et rejette la signature si $\mathbf{m'_R \notin M_R}$.
- B retrouve le message correctement signé par **A** en calculant: $\mathbf{m = R^{-1}(m'_R)}$.

1. *A Method for Obtaining Digital Signatures and Public Key Cryptosystems*. R.Rivest, A.Shamir and L.M. Adleman. Communications of tha ACM 21 (1978), 120-126

Signature RSA: Remarques

- La preuve de fonctionnement est identique à celle du procédé d'encryption (page 100). L'ordre d'exponentiation n'a pas d'influence puisque:

  $$\mathbf{ed \equiv de \equiv 1 \bmod \Phi(n)}$$

- Le procédé peut également être utilisé pour produire des signatures avec appendice avec les modifications suivantes:

  Signature:

  - A utilise une fonction de hachage **H** et calcule $\mathbf{m_h := H(m)}$.
  - A calcule la signature de m_h: $\mathbf{s := m_h^d \bmod n}$ et envoie le couple **(m,s) à B.**

  Vérification

  - B calcule $\mathbf{m'_h = s^e \bmod n}$ et **H(m)** et vérifie l'égalité $\mathbf{m'_h = H(m).}$
  - Si l'égalité est vérifiée, **B** accepte la signature **s** de **A** sur le message **M.**

- Le calcul de signature est plus lent que la vérification à cause de différence de taille entre l'exposant **d** (**taille(d)** ≈ **taille(Φ(n))** et **e**.
- Les risques et attaques mentionnés dans le procédé d'encryption (page 103) s'appliquent également pour la signature.
- Il convient de différencier les paires de clés d'encryption et de signature puisqu'elles nécessitent des politiques de stockage, sauvegarde et mise à jour distinctes.

Signatures "Aveugles" (*Blind Signatures)*

- Schéma inventé par Chaum ([Chau82][1]).
- Idée: **A** envoie une information à **B** pour signature. **B** retourne à **A** l'information signée. A partir de cette signature, **A** peut calculer la signature de **B** sur un autre message choisi à priori par **A**. Ceci permet à **A** d'avoir une signature de **B** sur un message que **B** n'a jamais vu (d'où le nom de signature aveugle...)
- En fait il s'agit d'une faille basée sur la propriété multiplicative de RSA (page 103) qui a été exploitée pour en faire un nouveau procédé de signature.
- Algorithme: Soit $\mathbf{S_B}$ la signature de RSA de **B** avec **(n,e)** et **d**, resp. les clés publiques et privées de **B**. Soit **k** un entier fixé avec **pgcd(n,k) = 1**:

  f: $\mathbf{Z_n \rightarrow Z_n}$ avec $\mathbf{f(m) = m \cdot k^e \bmod n}$; *blinding function*

  g: $\mathbf{Z_n \rightarrow Z_n}$ avec $\mathbf{g(m) = k^{-1} \cdot m \bmod n}$; *unblinding function*

  ce qui donne:

  $$\mathbf{g(S_B(f(m)) = g(S_B(mk^e \bmod n)) = g\,(m^d k \bmod n) = m^d \bmod n = S_B(m)} \quad \mathbf{(*)}$$

- Protocole:

  A → B: m' = f(m)

  A ← B: s' = $\mathbf{S_B}$(m')

  A calcule **g(s')** et obtient la signature souhaitée en utilisant **(*).**

1. [Chau82]: Chaum, D. *Blind Signatures for Untraceable Payments*. Crypto'82

Procédé de Signature de Rabin[1]

Génération des clés

- Chaque entité (**A)** crée une paire de clés (publique et privée) comme suit:
  - **A** génère deux nombres premiers aléatoires **p** et **q** de grande taille (*len* (pq) ≥ 1024).
  - **A** calcule **n := pq**.
  - La clé publique de **A** est **n** la clé privée de **A** est **(p,q)**.

Signature

- **A** calcule la fonction de redondance du message **m**: $\mathbf{m_R := R(m)}$.
- **A** utilise sa clé privée pour calculer la signature: $\mathbf{s := m_R^{1/2} \bmod n}$ en utilisant des algortihmes efficaces pour calculer des racines carrées **mod p** et **mod q.**
- **A** envoie **s** à **B** (**s** est **une des 4** racines carrées obtenues).

Vérification

- L'entité **B** obtient **n,** la clé publique authentique de **A**.
- B calcule $\mathbf{m'_R = s^2 \bmod n}$, vérifie $\mathbf{m'_R \in M_R}$ et rejette la signature si $\mathbf{m'_R \notin M_R}$.
- B retrouve le message correctement signé par **A** en calculant: $\mathbf{m = R^{-1}(m'_R)}$.

1. *Digitalized Signatures and Public Key Functions as Intractable as Factorization.* M.O.Rabin. MIT/LCS/TR 212. MIT Laboratory for Computer Science 1979.

Procédé de Signature d'ElGamal[1]

Génération des clés

- Chaque entité (**A**) crée une paire de clés (publique et privée) comme suit:
  - **A** génère un nombre premier **p** (**len(p)** ≥ 1024 **bits**) et un générateur α de $\mathbf{Z_p}^*$.
  - **A** génère un nombre aléatoire **a**, t.q. $\mathbf{1 \le a \le p-2}$ et calcule $\mathbf{y := \alpha^a \bmod p}$.
  - La clé publique de **A** est **(p, α, y)**, la clé privée de **A** est **a**.

Signature

- **A** utilise une fonction de hachage **H** et calcule $\mathbf{m_h := H(m).}$
- **A** génère un nombre aléatoire **k** $\mathbf{(1 \le k \le p-2}$ et **pgcd(k,p-1) = 1)** et calcule $\mathbf{k^{-1} \bmod (p-1)}$
- **A** calcule $\mathbf{r := \alpha^k \bmod p}$ et ensuite $\mathbf{s := k^{-1} (m_h - ar) \bmod (p-1)}$
- La signature de **A** sur le message **m** est le couple **(r,s)**.

Vérification

- L'entité **B** obtient **(p, α, $\alpha^a \bmod p$),** la clé publique authentique de **A**.
- **B** vérifie que $\mathbf{1 \le r \le p-2}$, sinon rejette la signature.
- **B** calcule $\mathbf{v_1 := y^r r^s \bmod p.}$
- **B** calcule **H(m)** et $\mathbf{v_2 := \alpha^{H(m)} \bmod p}$
- **B** accepte la signature ssi. $\mathbf{v_1 = v_2.}$

---

1. *A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logaithms*. T. ElGamal. Advances in Cryptology-Proceedings of CRYPTO 84 (LNCS 196), 10-18, 1985.

Signature ElGamal: Remarques

- Preuve que le schéma fonctionne: Si $\mathbf{s \equiv k^{-1} (m_h - ar) \bmod (p-1)}$, **on a que:**

  $$\mathbf{m_h \equiv (ar + ks) \bmod (p-1)} \text{ et}$$

  $$\mathbf{v_2 = \alpha^{H(m)} \bmod p}$$

  si, comme on souhaite montrer $\mathbf{m_h = H(m)}$, en réduisant les exposants **mod (p-1)** (page 93), on peur réécrire $\mathbf{v_2}$:

  $$\mathbf{v_2 \equiv \alpha^{ar+ks} \bmod p}$$

  D'autre part:

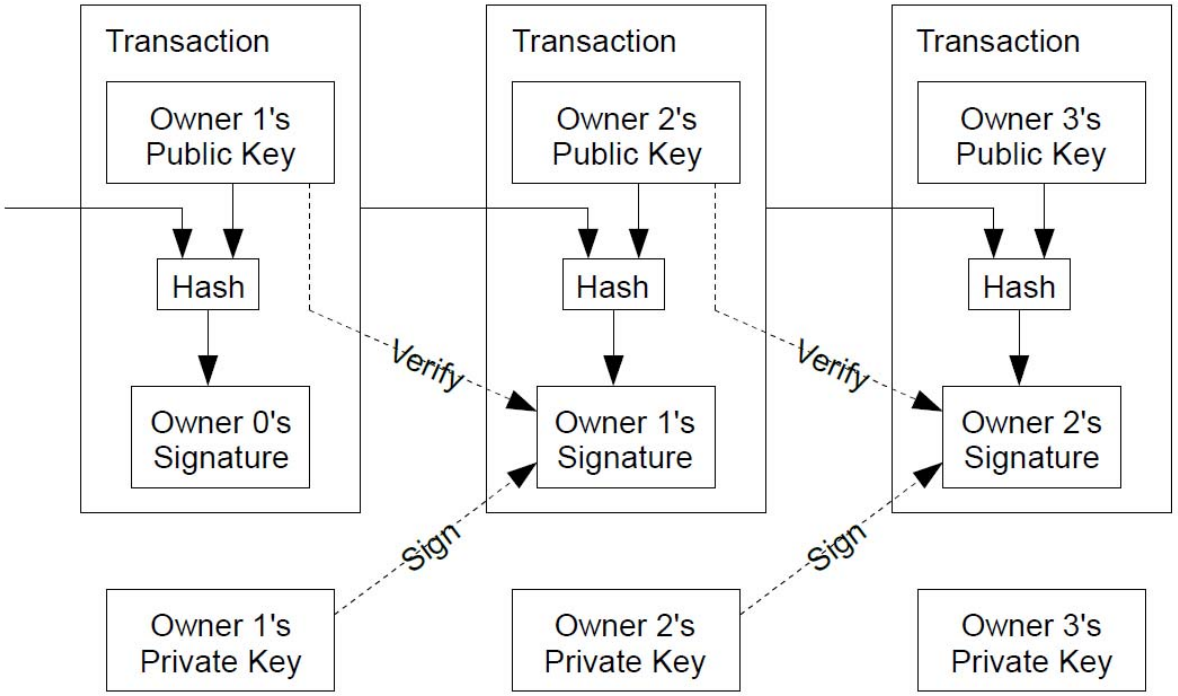
  $$\mathbf{v_1 = y^r r^s \equiv \alpha^{ar} \alpha^{ks} \equiv \alpha^{ar+ks} \bmod p} \qquad \mathbf{c.q.f.d.}$$

- Par construction, le schéma d'ElGamal fonctionne uniquement avec appendice (résultat de l'application d'une fonction de hachage). Le schéma de Nyberg-Rueppel [1] introduit une variation permettant la reconstitution du message.
- Le *Digital Signature Algorithm* (*DSA*), approuvé par le *US National Institute of Standards and Technology* est devenu le standard de signature le plus coramment utilisé. Il est construit sur la base d'un dérivé direct du schéma d'ElGamal avec la fonction de hachage *SHA-1* (page 139).

1. *Message Recovery for Signature Schemes Based on the Discrete Logarithm Problem*. K. Nyberg and R.Rueppel. Designs, Codes and Cryptography, 7 1996, 61-81.

Signatures Digitales: *Crypto-monnaies*

- La plupart des crypto-monnaies se basent sur la cryptographie asymétrique. Le ***bitcoin*** p.ex. utilise des signatures digitales pour authentifier ses transactions
- La dépense ou la transmission de bitcoins nécessite la signature avec la clé privée du détenteur (qui était à son tour le destinataire de la transaction précédente):



Source Image: *Bitcoin: A Peer-to-Peer Electronic Cash System.* Satoshi Nakamoto

- *Bitcoin* et *Ethereum* utilisent l'algorithme **ECDSA** (*Elliptic Curve Digital Signature Algorithm*) dérivé de algorithme de signature de ElGamal sur les courbes elliptiques dont la sécurité repose sur *ECDLP* (page 110).

Signatures Digitales: Schéma Récapitulatif[1]

Classe	Schéma	Message Recovery	Problème de base
Signatures Classiques	RSA	oui	RSAP
	Rabin	oui	SQROOTP
	ElGamal	non	DLP
	DSS	non	DLP
One-time Signatures	Lamport	non	dépend de la OWF
	Bos-Chaum	non	dépend de la OWF
Undeniable Signatures	Chaum-van Antwerpen	non	DLP
Fail-Stop Signatures	van Heyst-Pedersen	non	DLP
Blind Signatures	Chaum	oui	RSAP

1. Le fonctionnement des procédés de signature *One-time, Undeniable* et *Fail-Stop* peut être consulté dans [Men97].

Types d'attaques pour les SD

- Critères pour "casser" un schéma de signature digitale:
  - *Total Break*: Calculer la clé privé du signataire ou un algorithme efficace (polynomial) pour générer des signatures.
  - *Falsification sélective* (*selective forgery*). L'adversaire est capable de générer une signature valide pour un message (ou une classe de messages) fixé.
  - *Falsification existentielle* (*existential forgery*). L'adversaire est capable de forger une signature pour (au moins) un message (dont il n'a pas le contrôle).
- Attaques de base:
  - Attaques *key-only*: L'adversaire a seulement connaissance de la clé publique du signataire.
  - Attaques basées sur les messages: L'adversaire a accès à des signatures correspondantes à des:
    - *known-messages*
    - *chosen-messages*
    - *adaptive chosen-messages*

    Equivalents à des attaques *x-ciphertext* mais avec des messages !

Authentication
Outlook

- Data Origin Authentication
- Entity Authentication
  - Weak Authentication
  - Strong Authentication
  - Zero Knowledge Authentication

Authentification de l'origine des données

- 1) MAC avec une clé symétrique k connue de A et B:

  A → B: $\mathbf{X, MAC_k(X)}$

  Si B calcule de son coté $MAC_k(X)$ et obtient la même valeur => le message provient de A.

- 2) MDC + cryptage symétrique (clé k connue de A et B)

  A → B: $\mathbf{X, E_k(MDC(X))}$

  B calcule MDC(X) et puis $E_k(MDC(X))$. Si égal => message vient de A.

- 3) Comme 2) avec confidentialité de X en plus:

  A → B: $\mathbf{E_k(X,MDC(X))}$

- 4) MDC + signature digitale:

  A → B: $\mathbf{X, Sig_{priv\text{-}A}(MDC(X))}$

  B calcule MDC(X) et vérifie $Sig_{priv\text{-}A}(MDC(X))$ avec une copie authentique de `pub-A`. Si égalité => A est à l'origine du message.

  Cette solution offre en plus la non-répudiation d'origine.

- Ces protocoles simples n'offrent aucun support sur l'unicité ni sur l'actualité (*timeliness*) des messages reçus et sont exposés à des *replay attacks*! Ils nécessitent des mécanismes tenant compte du temps ou du contexte de la transaction (cf. authentification d'entités).

Authentification d'Entités: Introduction

- authentification d'entités (*entity authentication*), aussi appelé identification
- Objectifs d'un protocole d'identification robuste:
  - 1) Si A et B sont "honnêtes": si A est capable de s'authentifier auprès de B, B doit accepter l'identité de A.
  - 2) B ne peut pas réutiliser l'information remise par A pour s'identifier en tant que A auprès de C.
  - 3) La probabilité qu'une tierce entité C réussisse à se faire passer par A auprès de B est négligeable.
  - 4) Le point 3) reste vrai même si:
    - C a observé un grand nombre (polynomial) d'instances du protocole d'identification entre A et B
    - C a participé (ev. en se faisant passer par quelqu'un d'autre) à des exécutions précédentes du protocole d'identification auprès de A ou (non exclusif) B.
    - plusieurs instances du protocole (ev. initiées par C) peuvent s'exécuter simultanément sans compromettre le processus d'identification.
- Les protocoles d'*authentification faible* satisfont les points 1) et 3). Alors que les protocoles d'*authentification forte* satisfont (au moins partiellement) les points 2) et 4) en plus.

Authentification d'Entités: Introduction (II)

- Terminologie: pour la suite on appelle l'*utilisateur (A)*, l'entité devant prouver son identité (*claimant*) et le *système (B)* l'entité chargée de vérifier cette identité (*verifier*)
- Eléments de base pour l'authentification:
  - *something known*: passwords, PINs, clés privées ou secrètes, etc.
  - *something possessed*: passeport, carte à puces, générateurs de passwords, etc.
  - *something inherent to the human individual:* propriétés *biométriques* comme les empreintes digitales, la rétine, le code ADN, etc.
- **Authentification faible** (*weak authentication*): L'utilisateur présente un couple (*userid*, *password*) au système afin de s'identifier. Le *userid* étant l'identité prétendue et le *password* l'évidence corroborant.
- **Authentification forte** (*strong authentication*): Contrairement à l'authentification faible, le secret permettant de corroborer l'identité n'est pas révélé explicitement mais, plutôt, l'utilisateur fournit au système une preuve de possession de ce secret.
- Authentification par *zero knowledge:* Ce sont des protocoles d'authentification forte qui ont en plus la caractéristique de prouver l'identité de l'utilisateur sans dévoiler aucune information (ni même une piste...) sur le secret lui même. En d'autres mots, il s'agit de donner une preuve d'une assertion sans en révéler le moindre détail.

Attaques Dictionnaire (*Dictionnary Attacks*)

- Une **attaque dictionnaire** consiste à utiliser une base des données contenant des mots de dictionnaire d'une ou plusieurs langues (ainsi que des variantes) comme entrée à un système d'encryption ou de hachage afin d'obtenir des clés secrètes ou des passwords.
- Cette attaque est très efficace pour obtenir des mots de passe de mauvaise qualité même si dès nos jours il existent des bases de données de très grande taille contenant des variations de mots ainsi que de règles mnémotechniques complexes permettant de "casser" des mots de passe de plus forte entropie.
- Une attaque dictionnaire peut être montée:
  - en obtenant la base de données des mots de passe (encryptée ou *hashée*) du système d'authentification (système d'exploitation, réseau, etc.)
  - à partir d'un ou plusieurs échanges d'une instance d'authentification, suite à une attaque passive (observation de paquets réseau) p.ex.:

    A → B: `A` ; A envoie son identité
    A ← B: `R` ; R = un nombre aléatoire (*challenge*)
    A → B: $E_P(R)$; A encrypte R avec son password

    Le couple ($R, E_P(R)$) permet de monter une attaque dictionnaire *offline*.
- Les attaques dictionnaire sont normalement moins efficaces *online* car les systèmes d'exploitation limitent le nombre d'essais infructueux d'authentification.

Equivalence Plaintext (*Plaintext-Equivalence*)

- Une chaîne de données est dite *plaintext-equivalent* à un mot de passe si elle peut être utilisée pour obtenir le même niveau d'accès correspondant à l'utilisation du password.
- P.ex.: Si le système B stocke une liste de tous les mots de passe hashés de ses clients dans le procédé d'authentification suivant:

  A → B: `H(p)` ; A envoie à B le hash du password au système

  la chaîne d'information `H(p)` est *plaintext-equivalent* au mot de passe **p.**
  Ceci est équivalent à dire que l'application d'une fonction de hachage pour le stockage des passwords ne constitue pas une sécurité supplémentaire pour le système.
- En particulier, dans le système d'authentification classique d'UNIX (page 139) le hash du password stockée dans le fichier `/etc/passwd` n'est pas *plaintext-equivalent* au mot de passe car c'est `p` et non pas `H(p)` qui est échangé entre le client et le serveur.
- Cette propriété est essentielle car les bases de données des mots de passe sont normalement protégées par des mécanismes logiques qui sont souvent mis en évidence par des failles du système d'exploitation du serveur.
- Si ces bases de données centrales contiennent des mots de passe en clair ou des informations *plaintext-equivalent* à ces derniers, les conséquences en cas d'attaque sont dévastatrices.
- Le cas idéal est que les informations stockées par le serveur ne soient ni *plaintext-équivalent* aux passwords ni exposées à des attaques dictionnaire *offline*.

Authentification Faible

- Les systèmes d'authentification faible sont divisés en deux catégories principales:
  - *Password fixe*: Le password ne dépend pas du temps ni du nombre de fois que le protocole d'identification a été exécuté. Cette catégorie inclue les systèmes où le password est changé par décision de l'utilisateur ou par mesure de sécurité du système.
  - *Password variable*: La modification du password en fonction du temps et/ou du nombre d'exécutions fait partie du protocole d'identification.
- Techniques de stockage propres aux systèmes à password fixe:
  - stockage du password *en clair* dans un fichier protégé par les mécanismes de contrôle d'accès propres au système d'exploitation.
    Problèmes: failles dans le OS, privilèges du "super-user", *backups*, etc.
  - stockage du password encrypté ou après l'application d'une *one-way function* (éventuellement en rendant publique l'accès à ce fichier, cf. exemple UNIX).
    Problèmes: attaques *off-line*, ie. *guessing attacks*, *brute-force dictionary attacks*, *identification de collisions, etc.*
- Problème le plus grave du *password fixe*: il peut être rejoué après avoir écouté une instance d'identification sur un réseau non protégé !

Authentification Faible: Password fixe

- Techniques de protection des systèmes de password fixe:
  - Règles strictes de comportement concernant la création, le maintien et la mise à jour des passwords en tenant compte de la faible entropie des passwords choisis habituellement par les utilisateurs (cf. tableau ci-contre)

$\rightarrow c$ $\downarrow n$	26 (lowercase)	36 (lowercase alphanumeric)	62 (mixed case alphanumeric)	95 (keyboard characters)
5	23.5	25.9	29.8	32.9
6	28.2	31.0	35.7	39.4
7	32.9	36.2	41.7	46.0
8	37.6	41.4	47.6	52.6
9	42.3	46.5	53.6	59.1
10	47.0	51.7	59.5	65.7

Table 10.1: Bitsize of password space for various character combinations. The number of n-character passwords, given c choices per character, is c^n. The table gives the base-2 logarithm of this number of possible passwords.

$\rightarrow c$ $\downarrow n$	26 (lowercase)	36 (lowercase alphanumeric)	62 (mixed case alphanumeric)	95 (keyboard characters)
5	0.67 hr	3.4 hr	51 hr	430 hr
6	17 hr	120 hr	130 dy	4.7 yr
7	19 dy	180 dy	22 yr	440 yr
8	1.3 yr	18 yr	1400 yr	42000 yr
9	34 yr	640 yr	86000 yr	4.0×10^6 yr
10	890 yr	23000 yr	5.3×10^6 yr	3.8×10^8 yr

Table 10.2: Time required to search entire password space. The table gives the time T (in hours, days, or years) required to search or pre-compute over the entire specified spaces using a single processor (cf. Table 10.1). $T = c^n \cdot t \cdot y$, where t is the number of times the password mapping is iterated, and y the time per iteration, for $t = 25$, $y = 1/(125\ 000)$ sec. (This approximates the UNIX crypt command on a high-end PC performing DES at 1.0 Mbytes/s – see §10.2.3.)

(Source [Men97])

  - Ralentir le processus d'identification ainsi que limiter le nombre d'essais infructueux afin de contrer les "*on-line brute force attacks*".
  - *salting*: (cf. exemple UNIX)
  - Restreindre ou même éviter la diffusion des fichiers de mots de passe, même encryptés.

Authentification Faible: Password Variable

- Les deux techniques les plus connues d'identification par password variable sont les *one-time passwords* et les *générateurs (hardware) de nombres aléatoires*
- les *one-time passwords*: comme le schéma de *Lamport* (dont le *S-Key* est l'implantation la plus courante). Il fonctionne de la manière suivante:

Initialisation:

A génère un secret w
Une constante t (= nb. d'identifications ~1000) et une OWF H sont choisies
A→B: $w_t = H_t(w)$; H appliqué t fois à w.
B stocke: $w_{stored} := w_t$, $n := t-1$

Messages correspondants à l'identification $(t-n)^{ème}$:

A → B: A ; identité de A
A ← B: n ; itération courante pour A
A → B: $w_n = H_n(w)$
B teste: $H(w_n) == w_{stored}$. Si OK => $n := n - 1$ et $w_{stored} := w_n$

Fin: Quand n == 0, A choisit un nouveau w et on recommence...

- **Attaques**: Authentification de B **nécessaire**! Sinon: C se fait passer par B et:
  - obtient le mot de passe courant w_n et peut le rejouer (*pre-play attack*)
  - fournit un $n < n_{courant}$ et peut ainsi générer tous les $H_{m>n}(w_n)$ (*small n attack*)

Authentification Faible: Password Variable (II)

- Générateurs (hardware) de nombres aléatoires.
  - Il s'agit des cartes à puces qui génèrent périodiquement (~ tous les 30 ou 60 secs) des nombres différents servant à identifier (avec en plus, un *PIN* et des information sur l'identité de la personne) le détenteur de la carte.
  - La génération se fait à partir d'une clé secrète présente sur la carte et connue du système.
  - Le plus connu est *SecureId* fabriquée par *RSA Security.*
  - Il a été adopté par des nombreuses banques comme support d'authentification du *tele-banking* sur Internet.
  - Il est également exposé au *pre-play attack* mais le délai pour rejouer le password se limite à la fréquence de changement (30 ou 60 secs.).
- Conclusions authentification faible:
  - Les password fixes offrent un niveau de sécurité très réduit.
  - Les password variables constituent un pas important vers l'authentification forte mais nécessitent des précautions supplémentaires.

Authentification Forte: Solutions Symétriques

- Les protocoles d'authentification forte utilisent des techniques cryptographiques symétriques ou asymétriques. On commence par les symétriques...
- **Authentification unilatérale à clé symétrique partagée**:

A → B:	**A**	; A envoie son identité
A ← B:	**R**	; R = un nombre aléatoire (*challenge*)
A → B:	$\mathbf{E_{k\text{-}AB}(R)}$	; A encrypte R avec la clé partagée

B décrypte $E_{k\text{-}AB}(R)$ et identifie A s'il trouve R

- Remarques:
  - B doit s'assurer que le challenge R est aléatoire et ne doit pas le répéter.
  - Ce protocole constitue une amélioration remarquable par rapport à l'authentification par password car la variation des challenges empêche Eve de rejouer des parties du protocole.
  - *Eve* peut essayer un *off-line known-plaintext attack* à partir d'un nombre (qui reste normalement réduit) de couples $(R, E_{k\text{-}AB}(R))$ mais la plupart de systèmes de cryptage sont sûrs à cet égard (DES est vulnérable seulement à partir de 2^{47} paires)!
  - C peut se faire passer par B et choisir ses challenges R pour monter un *chosen-plaintext attack* (la vulnérabilité de DES à cet égard est aussi de 2^{47} mais d'autres systèmes de cryptage sont plus sensibles à ces attaques).

Authentification Forte: Solutions Symétriques (II)

- Remarques (suite):
  - C pourrait monter une attaque *Active Man-in-the Middle* en se faisant passer par B puisque B n'est pas authentifié) mais il doit convaincre A pour commencer le proto.
  - Un MDC: `H(k-AB,R)` ou un MAC: $H_{k\text{-}AB}(R)$ peuvent remplacer $E_{k\text{-}AB}(R)$ et accélérer l'identification.
  - Après l'identification initiale, un canal sûr (au moins authentifié) doit être établi à l'aide d'une protection cryptographique pour éviter que C puisse injecter des paquets en se faisant passer par A.
  - Les protocoles de ce type où une entité doit répondre en tenant compte d'un *challenge* proposé par l'autre s'appellent *challenge and response protocols* et sont la forme la plus répandue d'authentification forte.
- **Authentification unilatérale à clé symétrique partagée, 2ème variante**:

  A → B: **A,** $\mathbf{E_{k\text{-}AB}(timestamp)}$; horloges synchronisés entre A et B!
  - Avantage: un message en moins et protocole *stateless* **mais:**
  - la synchronisation d'horloges est difficile à obtenir dans la réalité et des "flottements" peuvent être exploités par un adversaire.
  - De plus, si on arrive à convaincre B "d'avancer sa montre", certaines instances d'identification passées peuvent redevenir valables.

Authentification Forte: Solutions Symétriques (III)

- **Authentification bilatérale à clé symétrique partagée (solution intuitive):**

  A → B: A, R_2

  A ← B: $R_1, E_{k\text{-}AB}(R_2)$

  A → B: $E_{k\text{-}AB}(R_1)$

- A première vue le protocole semble robuste **mais** observons ce que un adversaire C peut faire en démarrant deux processus d'identification:

  C → B: A, R_2 ; C prétend être A

  C ← B: $R_1, E_{k\text{-}AB}(R_2)$; B répond en se faisant passer par A

  à ce moment, C démarre une deuxième instance:

  C → B: A, R_1

  C ← B: $R_3, E_{k\text{-}AB}(R_1)$ **;** C ne peut plus poursuivre mais...

  complète avec succès la première instance d'identification avec:

  C → B: $E_{k\text{-}AB}(R_1)$; et c'est fait !

- Du fait que C renvoie à B le même R qu'il a reçu de lui, ce genre d'attaques s'appellent *reflection attacks*.
- Comme la clé est partagée, C aurait pu obtenir le même résultat (même plus discrètement) en exécutant la deuxième instance auprès de A (en prétendant être B)

Authentification Forte: Solutions Symétriques (IV)

- **Authentification bilatérale avec clé symétrique partagée (solution robuste):**

  (1) A → B: A, R_2

  (2) A ← B: $E_{k\text{-}AB}(R_1, R_2, A)$

  (3) A → B: $E_{k\text{-}AB}(R_2, R_1)$

- La présence de A dans (2) rajoute une sécurité supplémentaire au cas où les *reflection attacks* évidents ne sont pas détectés par le protocole. Autrement, si A lance une authentification avec celui qu'il croit B mais qui est en réalité C:

  A → C: A, R_2 **(*)**

  Alors C commence une nouvelle instance d'authentification avec A avec le même R_2:

  C → A: B, R_2 ; Si A ne voit pas R_2 comme réflexion évidente, alors il répond:

  C ← A: $E_{k\text{-}AB}(R_1, R_2)$; Comme dans (2) mais sans le 'A'

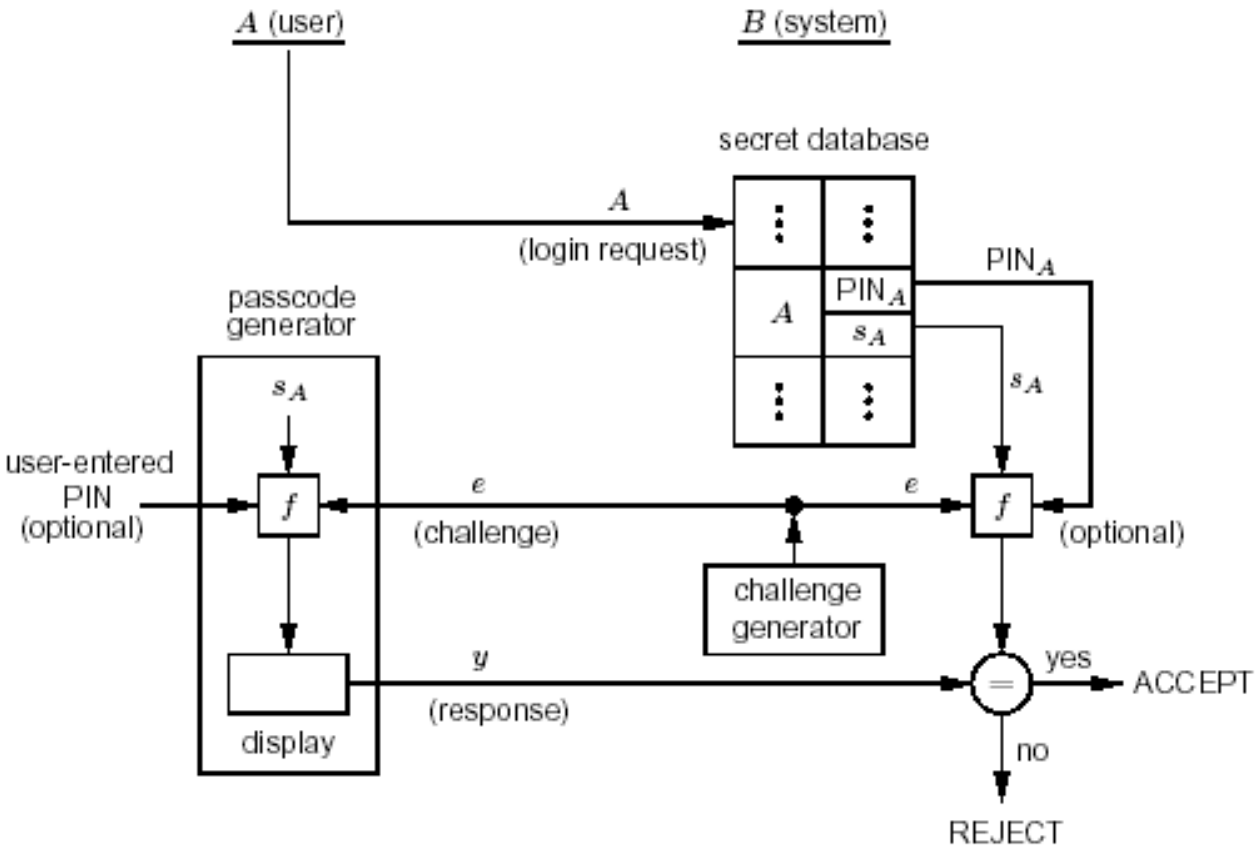
  ce qui est utilisé par C pour compléter son protocole **(*).** Cependant, si A répond avec B à l'intérieur du paquet comme recommandé dans le protocole:

  A → C: $E_{k\text{-}AB}(R_1, R_2, B)$

  ceci ne sera plus utilisable par C pour continuer **(*)** car il faudrait A à la place de B.

- A noter également que le fait d'inclure R_1 dans la partie encryptée protège également des dangers de *chosen plaintext attacks* de la solution précédente.

Solutions Symétriques: *Smartcards*



(Source [Men97])

Authentification Forte: Solutions Asymétriques

- **Authentification unilatérale à clé asymétrique (solution intuitive...)**:

  A → B: `A`
  A ← B: $\mathbf{E_{pub\text{-}A}(R)}$; B encrypte avec la clé publique de A
  A → B: `R` ; A retourne R après décryptage

- Remarques:
  - B doit connaître la clé authentique de A pour éviter des *man-in-the-middle attacks.*
  - **mais surtout:** B peut monter des *chosen-ciphertext attacks* (ie. B peut faire décrypter n'importe quoi à A!).
- **Authentification unilatérale avec clé asymétrique (solution robuste)**:
  Idée: structurer le texte encrypté avec pub-A et montrer que B connaît le *plaintext*:

  A → B: `A`
  A ← B: `H(R), B,` $\mathbf{E_{pub\text{-}A}(B,R)}$; H(R) témoigne du fait que B connaît R
  A décrypte $E_{pub\text{-}A}(B,R)$ et obtient B' et R'.
  A suspend le protocole si h(R') ≠ h(R) ou B' ≠ B, sinon:
  A → B: **R**
  B identifie A si coincidence avec le R initiale

- Un protocole dual peut être imaginé en utilisant la signature de A avec `priv-A` (au lieu de l'encryption avec `pub-A`) , mais les mêmes précautions concernant la structure s'appliquent pour éviter que A signe un message "mal intentionné" généré par B.

Authentification Forte: Solutions Asymétriques

- **Authentification bilatérale à clé asymétrique. Solution robuste due à Needham et Schroeder:**

  (1) A → B: $E_{pub\text{-}B}(r_1, A)$

  (2) A ← B: $E_{pub\text{-}A}(r_1, r_2)$

  (3) A → B: r_2

- A noter que la présence de A dans (1) démonte les *chosen ciphertext attacks*.
- Le protocole peut être renforcé en rajoutant un "témoin" $H(r_1)$ dans (1).

Remarques finales sur authentification classique

- L'authentification d'entités est un processus très complexe rempli de pièges inespérés.
- Certains protocoles comme celui proposé par l'ISO en 1988 pour l'authentification dans les répertoires distribués ont des failles très semblables à celles que nous avons mis en évidence ici.
- Par ailleurs, dans [Kau95], page 233, le protocole 9-9 a une faille non spécifiée et pour laquelle l'auteur ne donne pas de solution. A vous de la découvrir...
- Lorsque l'identification se fait dans le cadre d'une session, il est impératif que *tous les paquets* propres à la session soient authentifiés (p.ex. moyennant l'établissement d'un canal sûr avec l'établissement de clés de session).

Zero Knowledge Proofs: Définitions

- Problème avec les méthodes d'authentification "classiques": B (ou même un observateur) est en mesure d'obtenir des informations sur le secret détenu par A:
  - Dans les méthodes d'authentification faible (par *password*) c'est le secret dans son intégrité qui est dévoilé.
  - Dans les méthodes *challenge and response* classiques, B peut obtenir des couples [*plaintext* / *ciphertext*] pouvant servir à la cryptanalyse.
- *Définition*: Un protocole interactif est une **preuve de connaissance** (*proof of knowledge*) lorsqu'il a les deux caractéristiques suivantes:
  - **consistance** (*completeness*): si A et B sont deux entités "honnêtes", B accepte la preuve fournie par A.
  - **significativité** (*soundness*): Si une entité "malhonnête" C est capable de "tromper" B alors C détient le secret de A (ou une information *polynomialement* équivalente au secret). Ceci équivaut à exiger la possession du secret pour la réussite de la preuve.
- Une preuve de connaissance interactive est dite "*sans apport d'information*" (**zero knowledge interactive proof ou ZKIP**) si elle a, *en plus*, la propriété que A est capable de convaincre B sur un fait sans ne révéler *aucune information* sur le secret qu'elle possède.

Zero Knowledge Proofs: Définitions (II)

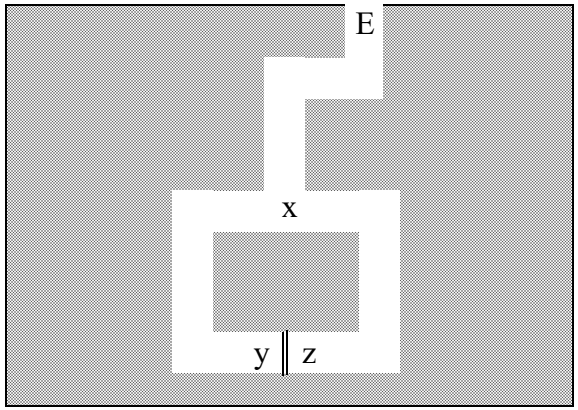
- Un protocole est une **ZKIP calculatoire** (*computational ZKIP*) si un observateur capable d'effectuer des tests probabilistes en temps polynomial n'est pas capable de distinguer une preuve authentique (ou A répond) d'une preuve simulée (p.ex. par un générateur aléatoire).
- Un protocole est une **ZKIP parfait** (*perfect ZKIP*) s'il n'existe aucune différence (au sens probabiliste) entre la vraie preuve et la preuve simulée. L'absence d'information dans la preuve est garantie par la *théorie de l'information* de Shannon et non pas par des critères calculatoires.
- Structure générique d'une ZKIP:

  (1) A → B: `témoin (witness)`
  (2) A ← B: `défi (challenge)`
  (3) A → B: `réponse (response)`

  - (1) A choisit un nombre aléatoire secret et envoie à B une preuve de possession de ce secret. Ceci constitue un engagement de la part de A et définit une classe de questions à laquelle A prétend savoir répondre.
  - (2) Le défi envoyé par B choisit (aléatoirement) une question dans cette classe.
  - (3) A répond (en utilisant son secret).
- Si nécessaire, le protocole est répété afin de réduire au maximum la probabilité qu'un "imposteur" devine "par chance" les réponses correctes.

ZKIP: Exemple Intuitif

- Cet exemple est décrit dans [Qui89][1].Admettons que A connaît un passage entre y et z (le secret).



(1) B se tient à l'entrée de la caverne au point E.
(2) A choisit une direction et se dirige vers les points y ou z (choix de témoin).
(3) Une fois A à l'intérieur de la caverne, B entre à son tour mais s'arrête au point x.
(4) B demande à A de se rendre au point x par la par la droite ou par la gauche (le défi).
(5) En utilisant le secret pour passer de y à z (ou réciproquement) si nécessaire, A obéit aux instructions de B.

Répéter les points 1 à 5 `n` fois. Si A ne connaît pas le secret, il a une probabilité de 2^{-n} de réussir à tromper B (de deviner "juste").

1.[Qui89]: Quisquater,J-J. et.al. *How to Explain Zero-Knowledge Protocols to Your Children*. Proceedings of Crypto'89 Conference. 1989.

ZKIP: Exemple Intuitif (II)

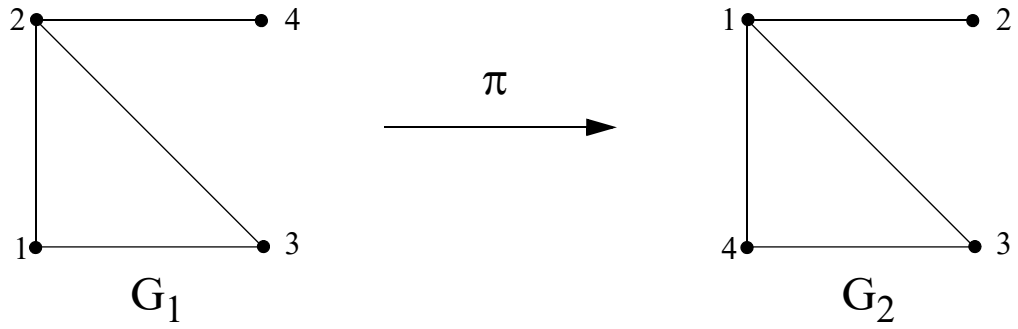
- Dans cet exemple, B constate que A peut traverser à volonté le passage `yz` mais n'obtient aucune information sur la manière de le faire même si le protocole est exécuté des millions de fois.
- Par ailleurs, B ne peut pas convaincre B' du fait que A connaît le secret (comme il aurait été le cas si A encryptait une information en utilisant une clé privée, p.ex.). B' pourrait suspecter A et B d'avoir convenu les séquences (droite/gauche)
- Ce genre de protocoles sont inspirés de la technique du "*cut and choose*" où A et B partagent équitablement une tarte en suivant les étapes suivantes:
  - A coupe la tarte.
  - B choisit un morceau.
  - A prend le morceau restant.
- Le premier ZKIP a été publié en 1985 par S. Goldwasser [Gol85][1]. L'application du paradigme du *cut and choose* aux protocoles cryptographiques est due à Rabin [Rab78][2].

---

1. [Gol85]: Goldwasser, S. et al. *The Knowledge Complexity of Interactive Proof Systems*. Proceedings of the 17th ACM Symposium on Theory of Computing, 1985.
2. [Rab78]: Rabin, M.O. *Digital Signatures*. Foundations of Secure Communications. New York, Academic Press, 1978.

ZKIP: Isomorphisme de Graphes

- Deux graphes $G_1 = (V_1,E_1)$ et $G_2=(V_2,E_2)$ sont *isomorphes* s'il existe une permutation π t.q. $\{u,v\} \in E_1$ ssi $\{\pi(u), \pi(v)\} \in E_2$.
- Exemple: $G_1 = (V, E_1)$ et $G_2=(V, E_2)$ avec $V = \{1,2,3,4\}$, $E_1= \{12,13,23,24\}$, et $E_2 = \{12,13,14,34\}$ sont isomorphes avec la permutation π: $G_1 \rightarrow G_2$, $\pi = \{4,1,3,2\}$:



G_1 G_2

- A partir d'un graphe G_1, on peut facilement (en temps polynomial) trouver une permutation π t.q.: $G_2 = \pi(G_1)$.
- Cependant, aucun algorithme polynomial n'est connu pour déterminer si deux graphes suffisamment grands (~1000 sommets) sont isomorphes (c.à.d. trouver la permutation π à partir des G_1 et G_2).

ZKIP: Isomorphisme de Graphes (II)

- Sur la base de l'isomorphisme des graphes, on peut construire une ZKIP comme suit:

  (**Initialisation**) A choisit un graphe G_1 suffisamment grand et invente une permutation π (le secret) lui permettant de calculer un deuxième graphe G_2 (G_1 et G_2 sont rendus publiques)

  (1) A → B: H ; A choisit une permutation aléatoire δ telle que δ: $G_2 \rightarrow H$ et
  ; envoie H à B (le témoin)

  (2) A ← B: i ; B choisit un entier $i \in \{1,2\}$ et l'envoie à A (le défi)

  (3) A → B: λ ; A calcule λ telle que $H = \lambda(G_i)$:
  ; Si $i = 2$: $\lambda := \delta$
  ; Si $i = 1$: $\lambda := \delta \circ \pi$

  (4) B contrôle si $H = \lambda(G_i)$ et accepte l'étape comme juste.

  (5) Répéter (1) à (4) un nombre de fois assez grand pour minimiser les risques de "deviner juste".

ZKIP: Isomorphisme de Graphes (III)

- Vérification des propriétés:
  - *Consistance*: Le protocole est accepté si A connaît le secret (ie. la permutation entre les deux graphes).
  - *Significativité*: Si C essaye de se faire passer par A sans connaître π, il pourra fixer un j et fournir une permutation correcte $\lambda(G_j)$ mais ne pourra pas trouver une permutation correcte pour les deux graphes. Il devra se contenter de deviner le *défi* fourni par B.
  - *Zero-Knowledge*: A réussit à convaincre B du fait que les deux graphes sont isomorphes mais n'apprend rien sur π; B ne voit qu'un graphe aléatoire H isomorphe à G_1 et G_2 ainsi qu'une permutation entre H et G_1 ou entre H et G_2.
  - *Zéro-Knowledge parfait:* ceci équivaut à dire que B pourrait générer des telles informations tout seul (à l'aide d'un générateur aléatoire et des calculs polynomiaux). On peut prouver (cf. [Sti95]) que les transcriptions fournies par le protocole ne peuvent se distinguer (d'un point de vue de distribution probabiliste) de celles produites par un simulateur (même en admettant que B "triche").
- L'utilisation du paradigme de l'isomorphisme de graphes dans les protocoles d'authentification reste relativement marginale dû à des problèmes d'efficacité d'implantation.

ZKIP: Algorithme de Fiat-Shamir

- But: Permettre à A de s'identifier en prouvant la connaissance d'un secret s (associé à A au moyen d'informations publiques authentiques) auprès de B sans lui révéler des informations sur s.
- Il s'agit d'un protocole qui sert comme base à des implantations réelles et efficaces.
- Algorithme:

(Initialisation):

(a) Un tierce de confiance, T choisit et publie un n t.q. n = pq et garde p et q secrets.

(b) A choisit un secret s avec $1 \leq s \leq n-1$ et $(s,n) = 1$, calcule $v = s^2 \bmod n$ et distribue v comme clé publique certifiée par T.

(1) A → B: $x = r^2 \bmod n$; A choisit un r aléatoire et envoie un témoin r^2

(2) A ← B: $e \in \{0,1\}$; B envoie son défi

(3) A → B: $y = r \cdot s^e \bmod n$; A calcule la réponse en utilisant le secret s.

B rejette la preuve si y = 0 (un imposteur pourrait fausser la preuve avec r = 0) et accepte la preuve si $y^2 \equiv x \cdot v^e \bmod n$.

Les étapes (1) à (3) sont répétées jusqu'à atteindre une marge de confiance suffisante.

ZKIP: Algorithme de Fiat-Shamir (II)

- Vérification des propriétés:
  - *Consistance*: Si A connaît s, le protocole accepte la preuve d'identification.
  - *Significativité*: Dans le cas simple, un imposteur pourrait seulement répondre à e = 0. Sinon, il pourrait choisir un r aléatoire et envoyer $x = r^2/v$ dans (1) et répondre au défi e = 1 avec une réponse correcte y = r. Dans le cas où e = 0, il devrait calculer la racine carré de x mod n (n composite de factorisation inconnue) ce qui est difficile par SQROOTP. La réussite de la preuve nécessite, donc, la possession du secret.
  - *Zéro Knowledge*: B ne peut obtenir aucune information sur s car lorsque e = 1, il est caché par un nombre aléatoire (*blinding factor*).
  - *Zéro-Knowledge parfait*: Les paires (x,y) obtenues de A peuvent également être simulées par B en choisissant un y aléatoire et un $x = y^2$ ou y^2/v mod n. On peut prouver que ces paires ont une distribution probabiliste identique à celles fournies par A (qui les calcule différemment!).
- A noter que, malgré cette dernière propriété, B est incapable de se faire passer par A auprès de B' car il ne peut pas prédire la valeurs des défis e.

ZKIP: Implantations Courantes

- **Feige-Fiat-Shamir (FSS)**:
  - Basé sur le protocole de Fiat-Shamir mais en utilisant des témoins et des défis multiples (des ensembles de k valeurs) à chaque itération; ce qui pour n itérations nous donne une probabilité de 2^{-nk} de deviner toutes les réponses.
- **Guillou-Quisquater (GQ)**:
  - Egalement basé sur Fiat-Shamir mais en augmentant le choix des défis ce qui diminue la probabilité de deviner sans augmenter le nombre d'instances transférées et d'étapes du protocole.
- **Schnorr**
  - Basé sur la difficulté de calculer des logarithmes discrets (DLP)
  - Il utilise également un domaine très grand de défis possibles ce qui lui permet de réaliser une identification en 3 échanges de messages seulement.
- Ces protocoles sont nettement plus efficaces que RSA et peuvent être implantés sur des supports à capacité de calcul réduite (*smart cards*).
- Il satisfont les propriétés de *consistance*, *significativité* mais la propriété *zero-knowledge* est parfois sacrifiée (comme dans le cas de *Schnorr*) pour augmenter l'efficacité.
- Pour une description détaillée de ces protocoles se référer à [Men97] ou à [Sti95].

ZKIP: Remarques Finales

- Les ZKIP offrent un très bon niveau de sécurité cryptographique. Ils permettent de procéder à des identification en minimisant les chances d'un imposteur hypothétique et, **surtout**, en protégeant les informations sécrètes des utilisateurs "honnêtes".
- En 1989 (SECURICOM'89) *Adi Shamir* disait à propos des ZKIP: "*I could go to a Mafia owned store a million successive times and they will still not be able to misrepresent themselves as me*"...
- Et pourtant ([Sch96]): A participe à une ZKIP avec C$\in$ Mafia; en même temps, D (complice de C) participe à une autre ZKIP où il prétend se faire passer par A auprès de B (un vérificateur "honnête").

(1)	A $\rightarrow$ C: t_1	; témoin que C fait suivre par liaison radio à D
(1)'	D $\rightarrow$ B: t_1	;
(2)'	D $\leftarrow$ B: d_1	; B envoie le défi à D; D le fait suivre à C...
(2)	A $\leftarrow$ C: d_1	; C reprend le défi dans son dialogue avec A
(3)	A $\rightarrow$ C: r_1	; la réponse en utilisant son secret, que C envoie à D
(3)'	D $\rightarrow$ B: r_1	; B accepte r_1 et ainsi de suite!

- Solutions:
  - Procéder à des identifications dans des chambres blindées (cage de *Faraday*)...
  - Utiliser des algorithmes de synchronisation forte pour éviter des échanges annexes.

Authentification: Récapitulation
Attaques et Protections

Attaque	Description	Protection
replay	rejouer une instance d'identification précédente	- *zero-knowledge* - *challenge and response* - *one-time password* (att. aux *pre-play* !)
known/chosen-plaintext	obtenir des couples plaintext /ciphertext	*zero-knowledge*
chosen-ciphertext	faire décrypter (ou signer) à A des informations soigneusement choisies	- *zero-knowledge* - *ch. & resp.* + témoin de connaissance + structure (redondance !)
reflection	répondre le même nombre qui a été reçu	- inclure l'entité cible dans les messages - asymétrie dans les messages
interleaving	utiliser des messages appartenant à plusieurs instances de protocoles simultanées	- inclure l'entité cible dans les messages - introduire un chaînage cryptographique entre les messages d'une même instance d'identification
collusion	connivence entre les intervenants	- cage de *Faraday* - synchronisation forte

Key Establishment Protocols (KEPs) Outlook

- Definitions
- Classification
- Properties
- *Key Agreement Protocols* (KAP):
  - Symmetric: *Authenticated Key Exchange* 2
  - Asymmetric: *Diffie-Hellman*, *Station to Station*, *Secure Remote Password*, *Off-The Record Protocol (OTR) and Signal Protocol*
- *Key Transport Protocols* (KTP):
  - Symmetric: *Shamir's No-key Protocol*, *Kerberos*
  - Mixed: *Encrypted Key Exchange*
  - Asymmetric: *Needham-Schroeder Protocol*
- SSL/TLS

Key Establishment protocols. Définitions

- Un protocole d'établissement de clés (*key establishment protocol* ou *KEP*) est celui qui met à disposition des entités impliquées un secret partagé (une clé) qui servira comme base pour des échanges cryptographiques ultérieurs.
- Les deux variantes des KEP sont les protocoles de transport de clé (*key transport protocol* ou *KTP*) et les protocoles de mise en accord (*key agreement protocol* ou *KAP*).
  - Un *key transport protocol* (KTP) est un mécanisme permettant à une entité de créer une clé secrète et de la transférer à son (ses) correspondant(s).
  - Un *key agreement protocol* (KAP) est un mécanisme permettant à deux (ou plusieurs) entités de dériver une clé à partir d'informations *propres à chaque entité*.
- *Key pré-distribution schemes* sont ceux où les clés utilisées sont entièrement déterminées à priori (p.ex. à partir des calculs initiaux).
- *Dynamic key establishment schemes (DKE)* sont ceux où les clés changent pour chaque exécution du protocole.

Key Establishment Protocols		
Key Agreement		**Key Transport**
symétrique + pré-dist.	symétrique + DKE	symétrique + DKE
asymétrique + pré-dist.	asymétrique + DKE	asymétrique + DKE

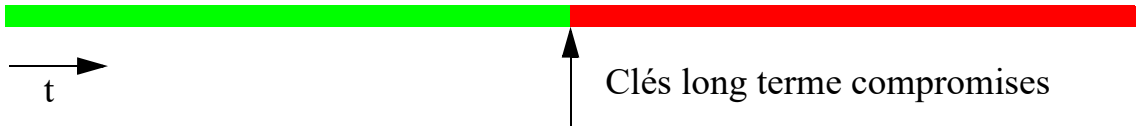
Key Establishment Protocols: Propriétés

- ***Implicit key authentication (****ou* ***key authentication)***: propriété par laquelle une entité est assurée que seul(s) son (ses) correspondant(s) peut (peuvent) accéder à une clé secrète. Cependant, ceci ne spécifie rien sur le fait de posséder effectivement la clé.
- ***Key confirmation***: propriété permettant à une entité d'être sûre que ses correspondants sont en possession des clés de session générées
- ***Explicit key authentication***: = *implicit key authentication + key confirmation.*
- Un ***authenticated KEP*** est un KEP capable de fournir *key authentication.*
- Attaques:
  - Une ***attaque passive*** est celle qui essaye de démonter un système cryptographique en se limitant à l'enregistrement et à l'analyse des échanges.
  - Une ***attaque active*** fait intervenir un adversaire qui modifie ou injecte des messages.
  - Un protocole est dit vulnérable à un ***known-key attack*** si lorsqu'une clé de session antérieure est compromise, il devient possible: (a) de compromettre par une attaque passive des clés futures et/ou (b) de monter des attaques actives visant l'usurpation d'identité.

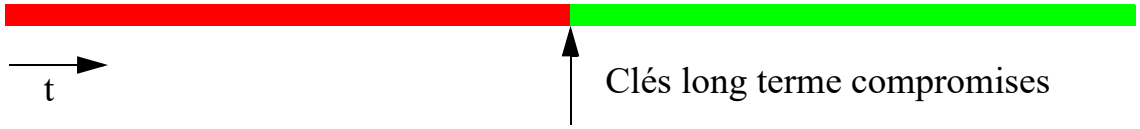
Key Establishment Protocols: Propriétés (II)

La plupart des protocoles modernes garantissent les propriétés suivantes:

- ***Perfect Forward Secrecy*** (PFS) est une caractéristique qui garantit la confidentialité des clés de sessions utilisées par le passé même si les clés long terme (par exemple la clé privé du destinataire) est compromise:



- ***Future Secrecy***: Le protocole garantit la sécurité des échanges ultérieurs (les clés des sessions futures sont protégées) même si les clés long terme sont compromises **par un attaquant passif**:



- ***Deniability / Repudiability*** (*répudiabilité*): A l'image des protocoles d'authentification *Zéro-Knowledge*, permet aux entités de garantir l'authentification des échanges sans apporter des informations qui permettraient de prouver à un tiers leur participation dans l'échange cryptographique.

KAP Symétrique avec Pré-distribution Cas Trivial

- Soit un nombre n d'utilisateurs avec un centre de distribution de clés (*key distribution center* ou *KDC*)
- On peut construire un KAP symétrique avec pré-distribution trivial de la façon suivante:

  (1) KDC génère `n(n-1)/2` clés différentes (une clé différente pour chaque couple d'utilisateurs).
  (2) KDC distribue en suite par un canal confidentiel et authentique les clés en donnant `n-1` clés à chaque utilisateur.
- Si KDC génère les clés de façon vraiment aléatoire, ce système est inconditionnellement sûr contre des complots d'utilisateurs (même en admettant que `n-2` utilisateurs complotent, ils ne pourraient pas trouver la clé des deux autres) par construction du protocole.
- Problème de ce protocole:
  - $O(n^2)$ en stockage de clés par le KDC.
  - O(n) en clés secrètes échangées pour chaque entité.

KAP Symétriques avec *Dynamic Key Establishment*

- Ces méthodes permettent aux entités impliqués de dériver des clés de courte durée (typiquement, des clés de session) à partir de secrets de longue durée qui, pour ces protocoles, sont des clés symétriques.
- Exemple intuitif:

  (Initialisation): A et B partagent une clé symétrique long terme **S**
  (1) **A** $\rightarrow$ **B:** $\mathbf{r_a}$; A génére un nb. aléatoire et l'envoie à B
  (2) **A** $\leftarrow$ **B:** $\mathbf{r_b}$; B fait de même
  A et B calculent la clé de session: $\mathbf{K := E_S(r_a \oplus r_b)}$

Propriétés:

- *Entity authentication:* NON: par construction du protocole, les r_i peuvent être envoyés par une entité quelconque.
- *Implicit key authentication*: OUI : seules les entités partageant la clé symétrique long terme **S** peuvent accéder à la clé de session **K**.
- *Key confirmation*: NON: les r_i étant aléatoires, ils peuvent être modifiés par un adversaire et empêcher A et B de se mettre d'accord sur la clé de session **K**. Ceci ne serait pas détecté par le protocole.
- *Perfect Forward Secrecy*: NON: si la clé long terme **S** est compromise, toutes les clés de session précédentes peuvent être facilement calculées par un adversaire qui aurait enregistré tous les échanges.

KAP Symétriques avec DKE
Authenticated Key Exchange Protocol 2 (AKEP2)[1]

(Init.): A et B partagent deux clés symétriques long terme **S** et **S'**. **S** est utilisé pour générer des **MACs $h_S()$** (afin de garantir l'intégrité et l'authentification d'entités) et **S'** pour la génération de la clé de session **K**.

(1) **A → B: r_a** ; A génère un nb. aléatoire et l'envoie à B

(2) **A ← B: T = (B,A,r_a,r_b), h_S(T)** ; B idem + identités + MAC de tout

(3) **A → B: (A, r_b), h_S(A,r_b)** ; A vérifie les identités et le r_a fournis par B ; ensuite, il envoie identité + r_b + MAC du tout.

La clé est calculée bilatéralement avec un **MAC** dédié **$h'_{S'}$()** : **K := $h'_{S'}$(r_b)**.

- *Entity authentication*: OUI mutuelle (fournie par les MACs).
- *Implicit key authentication*: OUI.
- *Key confirmation*: NON (pas d'évidence que la clé S' est connue du correspondant).
- *Perfect forward secrecy*: NON (si la clé S' est compromise, les clés de session K précédentes aussi).
- La clé dépend seulement de B (et de la clé long terme **S'**) mais le protocole peut être facilement modifié pour que la clé dépende aussi de A et en faire un "vrai" KAP.

1. [Bel93]: Bellare, M. et Rogaway, P. *Entity Authentication and Key Distribution*. Advances in Cryptography. CRYPTO'93.

KAP Asymétrique avec Pré-Distribution Diffie-Hellman

- Publié en 1976[1], il s'agit du précurseur des protocoles asymétriques.
- Il permet à deux entités qui ne se sont jamais rencontrées de construire une clé partagée en échangeant des messages sur un canal non confidentiel.
- Protocole:

  Initialisation: Un nb. premier p est généré et un générateur α de Z_p^*, t.q. $\alpha \in Z_{p-1}$. Les deux nombres sont rendus publiques.

  (1) **A $\rightarrow$ B:** α^x **mod p** ; A choisit un secret $x \in Z_{p-1}$ et envoie la partie publique

  (2) **A $\leftarrow$ B:** α^y **mod p** ; B choisit un secret $y \in Z_{p-1}$ et envoie la partie publique

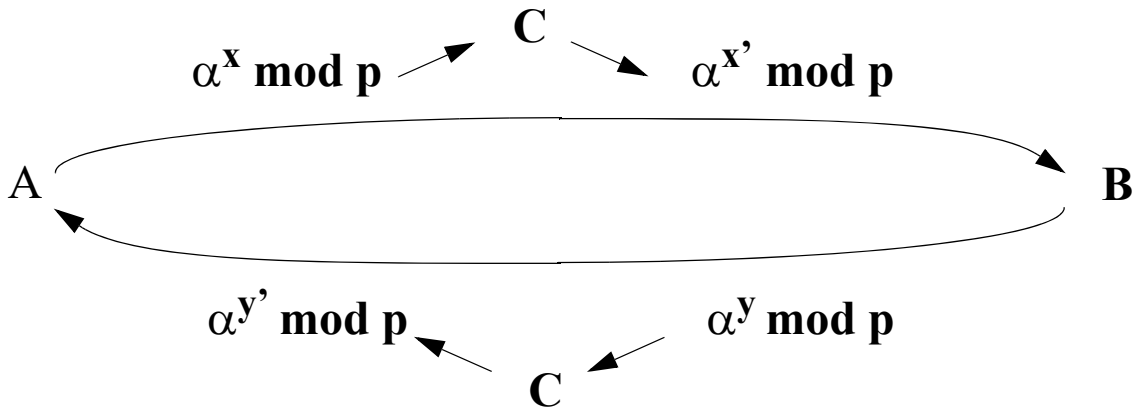
  A calcule la clé secrète: **K := $(\alpha^y)^x$ mod p** et B à son tour: **K := $(\alpha^x)^y$ mod p**

- La sécurité de ce schéma réside dans l'impossibilité de trouver α^{xy} mod p à partir de α^x mod p et α^y mod p. (*Diffie-Hellman Problem*: *DHP*).
- **Résultat prouvé**: DHP $\leq_p$ DLP.

1. [Dif76]: Diffie, W. and Hellman, M. *New Directions in Cryptography*. IEEE Transactions on Information Theory, 22 (1976), 644-654.

Diffie-Hellman (II)

- Diffie-Hellman est sûr (autant que DHP) contre des attaques passives. En d'autres mots, un adversaire qui se limite à voir passer des messages ne peut pas trouver la clé K.
- Ceci n'est cependant plus vrai pour des attaques actives; voyons ce que C peut faire en modifiant les messages:



- C échange des clés secrètes avec A et B, respectivement: $\alpha^{xy'}$ **mod p** et $\alpha^{x'y}$ **mod p** (C contrôle x' et y'). Si C ré-encrypte chaque paquet qu'il reçoit avec la clé publique correspondante, l'attaque se fera de manière transparente pour A et B.
- Cette attaque est appelée *Man in the Middle* (MIM) et s'applique à tous les protocoles asymétriques.
- Elle est due au manque d'authentification des clés publiques, ie. lorsque A "parle" à B, il doit utiliser la clé publique authentique de B.

Diffie-Hellman (III)

- Caractéristiques de Diffie-Hellman (non authentifié):
  - *Entity Authentication:* NON.
  - *Implicit key authentication*: NON (par l'attaque *MIM*).
  - *Key confirmation*: NON (dû au risque de *MIM*, A ne peut pas être sûr que B possède la clé secrète partagée).
- Génération de clés symétriques à partir d'une clé partagée Diffie-Hellman:
  - Les quantités manipulés dans DH (notamment K) sont de taille 512 - 1024 bits (suivant le nb. premier p utilisé).
  - Une approche intuitive pour générer des clés symétriques de petite taille (64 - 128 bits) serait de prendre un sous-ensemble de bits de la clé K.
  - Malheureusement, on peut prouver que les clés DH ne sont pas *bit secure* ce qui signifie que des sous ensembles de bits (notamment les *Least Significant Bits*) peuvent être calculés avec un effort non proportionnel à l'effort nécessaire à calculer la clé entière.
  - Pour générer des clés de manière sûre il est conseillé d'appliquer un MDC (comme SHA ou MD5) à *toute* la clé (ev. enchaîner l'application des MDCs pour obtenir des clés symétriques successives).
  - Cette méthode permet d'obtenir un KAP avec *Dynamic Key Establishment.*

Diffie-Hellman sur des Courbes Elliptiques

Génération des clés

- **A** et B choisissent une courbe elliptique $\mathbf{E_p}$ avec **p,** un nombre premier de grande taille (**len(p)** ~ 200 **bits**) et un point $\mathbf{P_o} \in \mathbf{E_p}$ de grand ordre (ev. un générateur de $\mathbf{E_p}$).
- **A** génère un nombre aléatoire **x**, t.q. $\mathbf{1 < x < p}$ et calcule la partie publique $\mathbf{xP_o}$ (multiplication par un scalaire sur $\mathbf{E_p}$, pour laquelle, il existe des algorithmes efficaces). **B** génère un nombre aléatoire **y** t.q. $\mathbf{1 < y < p}$ et calcule la partie publique $\mathbf{yP_o}$.
- Protocole d'échange de clés:

  (1) $\mathbf{A \rightarrow B: xP_o}$

  (2) $\mathbf{A \leftarrow B: yP_o}$
- Après l'échange A calcule $\mathbf{K_a := x\,(yP_o)}$ et B calcule à son tour: $\mathbf{K_b := y\,(xP_o)}$.
- La propriété commutative des opérations sur $\mathbf{E_p}$ garantit l´égalité: $\mathbf{K_a = K_b}$.
- La clé secrète partagée résulte d'un processus de sélection de bits de la clé $\mathbf{K_a}$ ou du résultat d'un MDC (fonction de hachage) appliqué à la clé $\mathbf{K_a}$.
- Il est également nécessaire d'authentifier les parties publiques échangées afin d'éviter les attaques *man-in-the middle* précédemment décrites.
- Les propriétés du protocole sont identiques au cas $\mathbf{Z_p^*}$ (page 193).

KAP Asymétrique avec DKE
Station to Station Protocol[1]

(Notation) S_A: Signature avec la clé privé de A.

(Initialisation):

(a) On choisit un nb. premier p et un générateur α de Z_p^* , t.q. $\alpha \in Z_{p-1}$. Les deux nombres sont rendus publiques (et éventuellement associés aux clés publiques des intervenants).

(b) Les intervenants ont accès aux copies authentiques des clés publiques des correspondants. Des certificats peuvent être échangés si besoin dans (2) et (3).

(1) A $\rightarrow$ B: $\alpha^{\mathbf{x}}$ **mod p** ; A génère un secret x et envoie la partie pub.

(2) A $\leftarrow$ B: $\alpha^{\mathbf{y}}$ **mod p,** $\mathbf{E_k(S_B(\alpha^x, \alpha^y))}$; B génére un secret y et calcule la clé:

k := $(\alpha^{\mathbf{x}})^{\mathbf{y}}$ **mod p** + signe et encrypte les p.pub.

(3) A $\rightarrow$ B: $\mathbf{E_k(S_A(\alpha^y, \alpha^x))}$; A décrypte en calculant **k :=** $(\alpha^{\mathbf{y}})^{\mathbf{x}}$ **mod p,**

; teste la signature de B et les parties publiques

; si OK, A signe + encrypte en inversant les

; parties publiques.

B décrypte et teste la signature de A sur les parties publiques. Si OK => FIN.

1. [Dif92]: Diffie, W. et al. *Authentication and Authenticated Key Exchanges*. Designs, Codes and Cryptography, 2 (1992).

Station to Station Protocol (II)

Caractéristiques:

- *Entity Authentication*: OUI mutuelle (fournie par les signatures).
- *Implicit key authentication*: OUI, les clés sont protégées par DHP. L'attaque MIM est rendue impossible par les signatures.
- *Key confirmation*: OUI, les deux entités prouvent la possession de la clé en encryptant des quantités avec.
- *Explicit key authentication:* OUI: *implicit key authentication* + *key confirmation*.
- *Perfect Forward Secrecy*: OUI. La seule clé à long terme est celle utilisée pour signature/ vérification. Si cette clé est compromise, les clés de session antérieures sont protégées par le fait qu'elles ne sont pas explicitement échangées mais plutôt calculées par DH.
- Evidemment, dès que la clé de signature est compromise (vol de clé privée), les propriétés énoncées ne sont plus vérifiées pour les échanges ultérieurs.
- Le protocole fournit en plus l'*anonymat* car l'identité des parties est protégée par k.
- Variante: Dans (2), calculer **sig := $S_B(\alpha^x, \alpha^y)$**, et envoyer: **(sig, h_k(sig))** plutôt que **$E_k(S_B(\alpha^x, \alpha^y))$.** Pareil pour (3) en observant les asymétries du protocole**.**
  Solution plus efficace car elle fait intervenir un MAC plutôt qu'un cryptage symétrique.
- Algorithme robuste et efficace choisi comme support de base pour la génération de clés dans *IPv6*.

Protocole ***Off-The-Record*** (***OTR***)

- Protocole conçu en 2004[1] dans le but d'offrir des services d'authentification et de confidentialité dans les échanges des messages (*instant messaging*) en préservant le caractère "*répudiable*" d'une conversation "*off the record*"
- Le protocole satisfait également les propriétés de ***PFS*** et ***Future Secrecy*** en cas de compromis des clés long terme.
- Il reprend les mêmes principes que le protocole *Station-to-Station* (page 197) en rajoutant aux signatures des paramètres DH une **authentification éphémère** via un MAC. Cette technique double est appelée **SIGMA**[2] (*SIGn-and-MAC*):
- Il utilise une fonction de dérivation de clés (*Key Derivation Function ou KDF*) pour générer une clé d'encryption (K_e) préservant la confidentialité des messages avec *AES CTR-mode* et une clé MAC (K_m) garantissant l'authenticité d'origine de ceux-ci.
- Chaque conversation implique un changement de clés (nouvel échange de paramètres DH) avec en plus **un échange en clair des clés MAC** (K_m) utilisées dans l'échange précédent pour ***garantir la répudiabilité*** !

---

1. N. Borisov, I. Goldberg, E. Brewer. *Off-the-Record Communication, or, Why Not To Use PGP.* Workshop on Privacy in the Electronic Society 2004.
2. H. Krawczyk. *SIGMA: The 'SIGn-and-MAc' Approach to Authenticated Diffie-Hellman and Its Use in the IKE Protocols.* CRYPTO 2003: Advances in Cryptology - CRYPTO 2003

Protocoles *OTR et Signal*

- Échanges schématiques du protocole OTR:

  (1) A → B: $\alpha^{\mathbf{x}}$ **mod p**; A génère un secret **x** et envoie la partie pub.

  (2) A ← B: $\alpha^{\mathbf{y}}$ **mod p,** $\mathbf{S_B(\alpha^x, \alpha^y))}$, $\mathbf{MAC_{Km}(B)}$

  B génère un secret **y**, calcule la clé de session $\mathbf{k := (\alpha^x)^y}$ **mod p** et signe les parties publiques DH. Il génère ensuite les clés K_e et K_m via la KDF: $\mathbf{(K_m, K_e) := KDF(k)}$

  (3) A → B: $\mathbf{S_A(\alpha^y, \alpha^x)}$, $\mathbf{MAC_{Km}(A)}$: A fait de même

  Les messages sont ensuite chiffrés avec la clé $\mathbf{K_e}$

- Il existe de nombreuses évolutions du protocole original OTR[1] ayant permis d'adresser des vulnérabilités et de rendre le protocole plus efficace.
- Le protocole **Signal**[2] est une évolution du protocole OTR qui cible la protection des échanges des messages dans les réseaux sociaux. Il utilise également des clés asymétriques et symétriques éphémères pour assurer la ***PFS***, la ***Future Secrecy*** et la ***repudiability*** avec des calculs DH sur des courbes elliptiques.
- **Signal** est utilisé pour protéger les plateformes de messagerie telles que *Whatsapp* et *Facebook Messenger* entre autres.

1.https://otr.cypherpunks.ca

2.https://signal.org (Open Whisper Systems)

Attaques Récentes sur Diffie-Hellman et la PFS

- En 2015 un groupe de chercheurs a publié[1] une série d'attaques sur le protocole TLS/SSL permettant de:
  - Effectuer un *downgrade* via une attaque active appelée *Logjam* moyennant laquelle un *man-in-the-middle* réussit à diminuer à 512 bits la taille du groupe Diffie-Hellman sur lequel s'effectue l'établissement de la clé secrète partagée.
  - Calculer ensuite les logarithmes discrets de $\alpha^{\mathbf{x}}$ **mod p** et de $\alpha^{\mathbf{y}}$ **mod p** avec la technique *Number Field Sieve*.
- A partir d'un groupe basé sur un nombre premier **p** fixé, ils effectuent une phase de pré-calcul d'une durée approximative d'**une semaine**.
- Une fois cette phase initiale terminée, les calculs des logarithmes individuels ne prennent qu'**une minute**!
- Une constatation statistique montre qu'un pourcentage significatif des serveurs se basent sur le même groupe (même premier p) ce qui permet d'utiliser la même phase de pré-calcul pour compromettre plusieurs serveurs.
- Une des conclusions de cette recherche est que des acteurs majeurs avec des ressources étatiques seraient capable à ce jour de démonter la *PFS* lorsque celle-ci est basé sur des groupes (très répandus à ce jour...) de 1024 bits.

1. [Adr15] Adrian, A. et al. *Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice*. 22nd ACM Conference on Computer and Communications Security, CSS 2015.

KAP Asymétrique avec DKE
Secure Remote Password protocol[1]

(a) Soit m un *safe prime* avec **m := 2p+1** et **p** premier

(b) Soit α un générateur de Z_p^*, t.q. $\alpha \in Z_{p-1}$

(c) Soit **P** le password de A et **x := H(P)** avec **H** une CRHF.

(d) B garde dans sa base des mots de passe le *vérificateur* **v** := $\alpha^{\mathbf{x}}$ **mod m.**

(1) A $\rightarrow$ B: δ **:=** $\alpha^{\mathbf{r}}$ **mod m** ; A génère un nombre aléatoire secret **r**

(2) A $\leftarrow$ B: λ **:= (v +** $\alpha^{\mathbf{t}}$**) mod m, u** ; B génère un nombre aléatoire secret **t** et un deuxième nombre aléatoire **u**

A calcule la clé symétrique**: k := (** λ **– v)**$^{\mathbf{r+ux}}$ **mod m**

B calcule la clé symétrique**: k :=** $(\delta \mathbf{v}^{\mathbf{u}})^{\mathbf{t}}$ **mod m**

A et B prouvent la connaissance de **k** (*key confirmation*) lors d'un échange ultérieur.

- **SRP** protège les mots de passe des attaques dictionnaire.
- B ne stocke pas les passwords mais des valeurs de vérification (***verifier-based***).
- **SRP** satisfait également toutes les propriétés propres aux KEP et est inclus dans des nombreux standards (SSL/TLS, EAP, etc.).

1.[Wu00]: Thomas Wu.*The SRP Authentication and Key Exchange System*. Request for Comments: 2945. Septembre 2000.

Key Transport Protocol Symétrique
Cas trivial

(Init.) A et B partagent une clé symétrique long terme **S**

(1) **A → B:$E_S(r_a)$** ; A génère un nb. aléatoire et l'encrypte avec k

La clé de session utilisée par les deux entités est **K := r_a**.

Propriétés:

- *Entity Authentication*: NON.
- *Implicit Key Authentication*: OUI (seul A et B ont accès à la clé).
- *Key Confirmation*: NON. B ne peut pas être sûr que A possède la clé car r_a est un nombre aléatoire. En rajoutant de la redondance (p.ex. l'identité de B), B peut obtenir *key confirmation* unilatérale (et donc, *explicit key authentication*):

  (1)': **A →B: $E_s(B, r_a)$**
- *Perfect Forward Secrecy*: NON.
- Si, de plus, B ne peut pas juger l'actualité (*freshness*) de (1) à partir du seul r_a, il peut demander à A d'inclure un *timestamp* à condition d'avoir des horloges synchronisés:

  (1)'': **A →B: $E_s(B, t_a, r_a)$**

KTP Symétrique: *Shamir's No-key Protocol*[1]

Rappel Théorie des nombres: Si p premier et $r \equiv t \bmod p-1$ alors $a^r \equiv a^t \bmod p \ \forall\ a \in Z$ et donc: $rr^{-1} \equiv 1 \bmod p-1$ implique $a^{rr-1} \equiv a \bmod p$.

(Init.) (a) Choisir et publier un nb. premier p pour lequel il est difficile (par DLP) de calculer les logarithmes discrets dans Z_p.
(b) A (resp. B) génére un nombre secret a (resp. b), t.q $\{a,b\} \in Z_{p-1}$ et $(a,p-1) = 1$ et $(b,p-1) = 1$ (pour que les inverses existent).
(c) Pour la suite, A pré-calcule $a^{-1} \bmod p-1$ et B pré-calcule $b^{-1} \bmod p-1$

(1): **A → B: $K^a \bmod p$** ; A choisit une clé $K \in Z_p$ et la cache avec ^a

(2): **A ← B: $(K^a)^b \bmod p$** ; B exponentie à son tour avec b

(3): **A → B: $(K^{ab})^{a-1} \bmod p$** ; A défait l'exponentiation avec $a^{-1} \bmod p-1$
; mais la clé reste protégée par ^b

B n'a plus qu'à calculer K en exponentiant avec $b^{-1} \bmod p-1$.

- Ce protocole est l'équivalent de Diffie-Hellman en *Key Transport* (dans DH la clé n'est pas transportée mais calculée bilatéralement). Il souffre donc des mêmes problèmes (notamment *Man in the Middle*) que ce dernier.

1.[Mas92]: Massey, J.L. *Contemporary Cryptology: An Introduction in Contemporary Cryptology: The Science of Information Integrity.* G.J Simmons, ed., IEEE Press, 1992.pp 1-39.

Key Transport Protocol Asymétrique
Needham-Schroeder Public Key Protocol[1]

(Notation): $E_{pubE}(X)$ signifie *encrypter avec la clé publique de l'entité E*.

(Init): A et B possèdent une copie authentique (ev. un certificat) de la clé publique de l'autre.

(1) **A → B: $E_{pubB}(k_1,A)$** ; A génère un nb. aléatoire k_1 + A + Encrypt

(2) **A ←B: $E_{pubA}(k_1,k_2)$** ; B idem pour k_2+ concat avec k_1 + Encrypt

(3) **A → B: $E_{pubB}(k_2)$** ; A vérifie si k_1 coïncide, si oui, encrypt k_2

; B vérifie si k_2 coïncide avec (2)

La clés est générée à l'aide d'une fonction de hachage cryptographique: **$K := H(k_1,k_2)$**

Caractéristiques:

- *Entity Authentication* + *implicit key authentication* + *key confirmation*: OUI.
- *Perfect forward secrecy:* NON: Les clés sont entièrement déterminées par les quantités échangées.
- Un protocole semblable (seul (3) change) peut être utilisé pour l'authentification d'entités (cf. chapitre authentification).

1.[Nee78]: Needham, R.M. et Schroeder, M.D. *Using Encryption for Authentication in a Large Network of Computers*. Communications of the ACM, 21, 1978.

KTP mixte: *Encrypted Key Exchange* (EKE)[1]

- Ce protocole fait intervenir des schémas symétriques et asymétriques afin de minimiser le risque de cryptanalyse par *attaque dictionnaire* inhérents aux systèmes symétriques.

  (Init.): A et B partagent un secret symétrique p (*password*).

  (1) **A → B:** $\mathbf{A, E_p(pub_A)}$; A génère une paire de clés pub/priv. et envoie
  ; la partie publique à B encrypté avec p.
  (2) **A ← B:** $\mathbf{E_p(E_{pub_A}(k))}$; B génére une clé de session k et l'envoie encryptée.
  (3) **A → B:** $\mathbf{E_k(r_a)}$; A génére un nb. aléatoire et l'envoie encrypté avec k.
  (4) **A ← B:** $\mathbf{E_k(r_a,r_b)}$; B génére r_b et l'envoie avec r_a crypté avec k.
  (5) **A → B:** $\mathbf{E_k(r_b)}$; Confirmation de la part de A. Si r_b = OK => FIN.

- (1) et (2) sont responsables du *key transport*; (3) à (5) du *key confirmation.*
- Ce protocole est robuste même si le *password* p partagé entre A et B est de mauvaise qualité. En effet, *Eve* ne peut pas essayer de deviner sans "casser" aussi l'algorithme asymétrique.
- *Entity Authentication* + *implicit key authentication* + *key confirmation*: OUI.
- *Perfect forward secrecy:* OUI si la paire $pub_A/priv_A$ est régénérée à chaque instance du protocole. NON si $pub_A/priv_A$ est une clé de longue durée.

1. [Bell93]: Bellovin, S.M. and Meritt, M. *Encrypted Key Exchange: Password-Based Protocols Secure Against Dictionary Attacks*. 1992 IEEE CS Conf. on Research in Security and Privacy 1992.

KTP symétrique avec *Key Distribution Center Needham-Schroeder Symétrique*[1]

(Notation): On appelle T, le *Key Distribution Center.*
(Init.): A et T partagent la clé symétrique K_{AT}. B et T partagent K_{BT}.

(1) **A → T:** **A, B, r_a** ; A génére un nb. aléatoire r_a et l'envoie à T avec les ident.

(2) **A ← T:** **$E_{K_{AT}}(r_a, B, k_{AB}, E_{K_{BT}}(k_{AB}, A))$** ; T génére k_{AB} et l'envoie encryptée.

(3) **A → B:** **$E_{K_{BT}}(k_{AB}, A)$** ; A forwarde le paquet à B.

(4) **A ← B:** **$E_{k_{AB}}(r_b)$** ; confirmation de B en utilisant k_{AB} et un nb. aléatoire r_b

(5) **A → B:** **$E_{k_{AB}}(r_b - 1)$** ; confirmation de A

- *Entity Authentication*:
  - A auprès de B: OUI.
  - B auprès de A: NON: A n'a jamais vu r_b (il pourrait s'agir de $E_{k'}(r_b')$).
- *Implicit Key Authentication*: OUI (les clés sont toujours protégées par K_{AT} et K_{BT}). Cependant, en cas de *known-key attack* (cf. page suivante), ceci n'est plus vérifié pour B.
- *Key Confirmation*: Seul B obtient l'assurance que A possède la clé à cause de la faille décrite dans *entity authentication.*

1.[Nee78]: Needham, R.M. et Schroeder, M.D. *Using Encryption for Authentication in a Large Network of Computers*. Communications of the ACM, 21, 1978.

KTP symétrique avec *Key Distribution Center* *Needham-Schroeder Symétrique (II)*

- *Perfect Forward Secrecy*: NON. Si une des deux clés K_{AT} ou K_{BT} est compromise, les clés de session k deviennent immédiatement visibles.
- Solution pour obtenir *key confirmation* et *entity authentication* mutuelles:

  Remplacer: (3) et (4) par:

  (3')**A → B:** $\mathbf{E_{kAB}(r_a'), E_{KBT}(k_{AB}, A)}$

  (4')**A ← B:** $\mathbf{E_{kAB}(r_a'\text{-}1, r_b)}$

  Pour autant que les r_i soient soigneusement contrôlés par les intervenants.

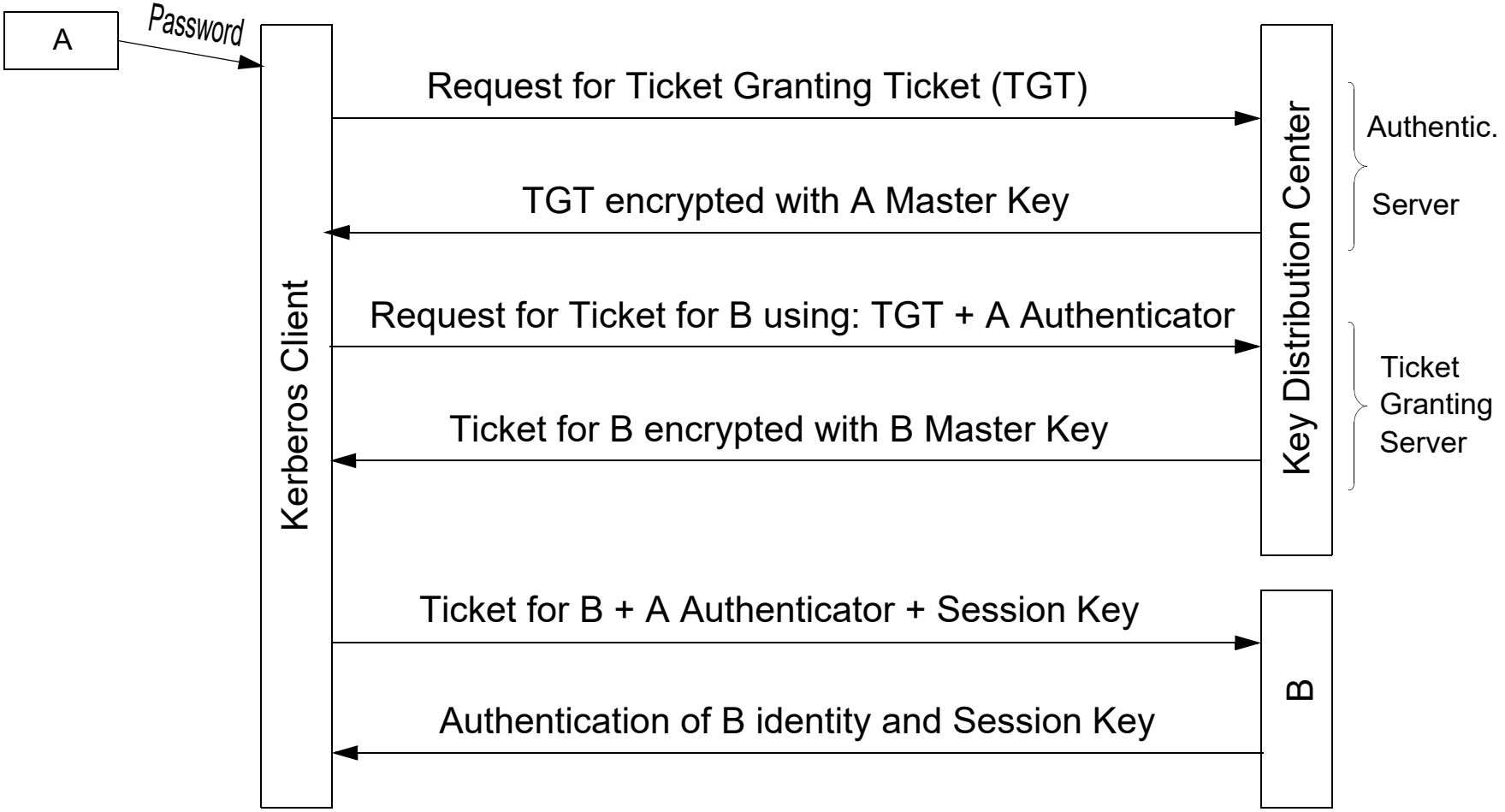
  Cependant: attention aux *reflection attacks* (cf. authentification d'entités)!
- **Problème**: A peut rejouer (3) autant des fois qu'il le souhaite, sans aucun contrôle de la part de B. Ce problème s'aggrave si une vielle clé k est compromise:
- ***Vulnérable au known-key attack***: Si une clé de session k déjà utilisée est obtenue par un adversaire C, il peut sans difficulté la faire accepter par B en rejouant (3) et en calculant le *challenge* envoyé par B dans (5). Dans ce cas, les propriétés *entity authentication*, *implicit key authentication* et *key confirmation* de A auprès de B sont aussi compromises.
- Solution: Rajouter un *timestamp* dans (3) témoignant de l'actualité des échanges:

  (3'') **A → B:** $\mathbf{E_{KBT}(k_{AB}, A, t)}$ **(**c'est la solution adoptée par Kerberos)

KTP symétrique avec *Key Distribution Center Kerberos*

- Kerberos est un protocole permettant l'authentification d'entités et la distribution de clés à l'intérieur dans un réseau d'utilisateurs.
- A l'origine, Kerberos était conçu comme solution de remplacement pour remédier aux problèmes d'insécurité (authentification faible, transactions en clair, etc.) propres aux environnements UNIX.
- Kerberos fut crée à MIT comme partie intégrante du projet ATHENA.
- Il est basé sur le protocole de *Needham-Schroeder* symétrique avec notamment la correction de quelques failles du protocole et l'inclusion de *timestamps*.
- Les trois premières versions étaient instables. La version 4 a eu un succès considérable aussi bien dans les environnements industriel qu'académique et reste prédominante. La version 5, bien qu'étant plus sûre et mieux structurée, est plus complexe et moins performante, ce qui a ralenti son déploiement.
- Kerberos définit également un mode de collaboration entre domaines appartenant à des autorités administratives distinctes (les *realms*). Ceci permet à des utilisateurs d'un domaine d'utiliser des ressources d'un autre domaine "sans sortir" de l'environnement sécurisé de Kerberos.
- Pour des transactions *inter-realm*, la cryptographie symétrique constitue un obstacle significatif car nécessite des canaux confidentiels pour la pré-distribution des clés.

Kerberos: Schéma Simplifié



[Kau95] **contient une description complète de Kerberos**

Kerberos Version 5

(Notation): **- A** et **B** veulent établir une transaction sécurisée; dans l'environnement Kerberos, il s'agit normalement d'un client et d'un serveur fournissant des services.
- Le *KDC* de Kerberos est subdivisé en deux entités fonctionnelles: l'*Authentication Server* (**AS**) et le *Ticket Granting Server* (**TGS**). Les deux accèdent à la *BdD passwords*.
- Les $r_a^{(n)}$ sont des nbs. aléatoires, t est un *timestamp*, t_1 et t_2 indiquent une fenêtre de validité de temps.

(Initialisation): A et B partagent une clé secrète avec AS, soient: K_A et K_B (pour les clients, il s'agit d'une OWF du *password*). TGS a également une clé secrète K_T.

(1) **A → AS:** **A, TGS, r_a**
(2) **A ← AS:** **$E_{KA}(k_{AT},r_a)$, $Ticket_{AT} := E_{KT}(A,TGS,t_1,t_2,k_{AT})$** ; AS génére k_{AT}
(3) **A → TGS:** **$Authenticator_{AT} := E_{kAT}(A,t)$, $Ticket_{AT}$, B, r_a'**
(4) **A ← TGS:** **$E_{kAT}(k_{AB},r_a')$, $Ticket_{AB} := E_{KB}(A,B,t_1,t_2,k_{AB})$** ; TGS génére k_{AB}
(5) **A → B:** **$Authenticator_{AB} := E_{kAB}(A,t)$, $Ticket_{AB}$, r_a", [request]**
(6) **A ← B:** **$E_{KAB}(r_a")$, [response]** ; [request] et [response] ev. cryptés avec k_{AB}

(1) + (2): **Demande de ticket pour TGS**
(3) + (4): **Demande de ticket pour B**
(5) + (6): **Authentification et établissement de clé entre A et B.**

Caractéristiques de Kerberos

- *Entity Authentication*: OUI, de toutes les entités impliquées.
- I*mplicit key authentication*: OUI: toutes les clés générées sont protégées par des clés partagées entre le AS et tous les participants.
- K*ey confirmation*:
  - Entre A et AS: NON: AS n'a pas de preuve que A possède la clé K_A.
  - Entre A et TGS: OUI pour k_{AT} (des quantités redondantes encryptées avec k_{AT} sont échangées entre A et TGS); NON pour k_{AB} (TGS n'a pas de preuve de la part de A)
  - Entre A et B: OUI: échange des quantités redondantes encryptées avec k_{AB}.
- *Perfect forward secrecy:* NON: Toutes les clés sont explicitement transférées.

Problèmes:

- Les clés initiales (comme K_A) dépendent (directement) des *passwords* choisis par les utilisateurs. Ceci rend le protocole vulnérable à des *vols de password* ou à des:
- *Password guessing attacks*: $\mathbf{E_{K_A}(k_{AT},r_a)}$ dans (2) aide à casser le password de A. Solution: Pré-authentification dans **(1):** $\mathbf{E_{K_A}(t)}$ avec t = *timestamp* (optionnelle dans *v5*).
- La fenêtre de validité d'un *ticket* peut conduire à des *replay attacks* si les $r_a^{(n)}$ ne sont pas correctement contrôlés par les intervenants.
- La synchronisation d'horloges est *nécessaire*! Ceci n'est pas toujours facile dans des environnements hétérogènes.

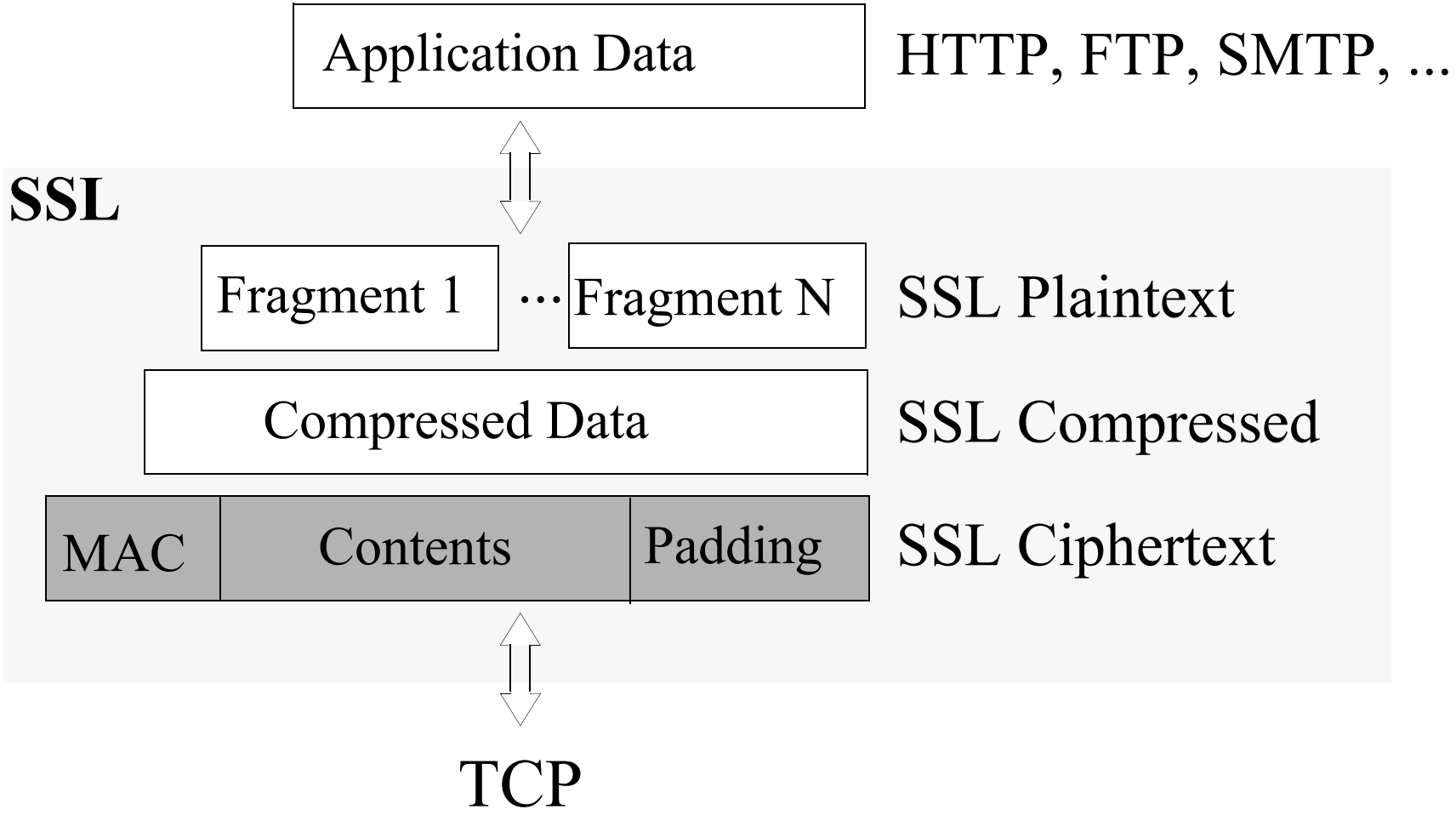
Secure Socket Layer (SSL)/ *Transport Level Security* (TLS)

- Se situe entre la couche transport (TCP) et les protocoles de la couche application (non seulement HTTP mais également SMTP, FTP, etc. !)
- Il s'agit d'un *Meta Protocole d'établissement de clés* hautement paramétrable permettant des nombreux modes de fonctionnement et des options de négociation.
- Offre des services de confidentialité, intégrité, authentification du flot de données, et identification du serveur (et accessoirement du client)
- Utilise les familles d'algorithmes suivants:
  - Cryptographie publique (RSA, *Diffie-Hellmann*, DSA, etc.) pour l'échange de clés symétriques
  - MACs pour l'authentification du flot de données
  - Cryptographie symétrique (DES, IDEA, AES, etc.) pour l'encryption du flot de données
- L'intervention des *CAs* pour *certifier* l'association entre entités et clés publiques est vivement recommandée... mais pas indispensable !
- *Entity authentication* par certificats (serveur et client optionnelle), *Implicit Key Authentication* et *Key Confirmation* sont garanties. La *Perfect Forward Secrecy* depend du protocole choisi pour l'échange de clés.

SSL/TLS Aperçu

- SSL est une “mini-pile” de protocoles avec des fonctionnalités des couches session, présentation et application.
- SSL est constitué de trois blocs fondamentaux:
  - *SSL record protocol* permettant l’encapsulation des protocoles de plus haut niveau au dessus de TCP (fragmentation + compression + encryption)
  - *SSL handshake protocol* chargé de l’authentification des intervenants et de la négociation des paramètres d’encryption
  - *SSL state machine*. Contrairement à HTTP, SSL est un protocole à états *(stateful)*, il nécessite, donc, un ensemble de variables qui déterminent l’état d’une session et d’une connexion

SSL/TLS *Record Protocol*



MAC = Message Authentication Code: Permet d'assurer l'intégrité et l'authenticité du paquet

SSL/TLS *State Variables*

Par session

- **session identifier**
- **peer certificate**
- **compression method**
- **cipher specification**
  encryption algo. +
  authentication algo. +
  crypto attributes
- **master secret**
  secret shared between client and server
- is **resumable**
  indicates if new connections may be initiated under this session

Par connexion

- **client/server random**
  byte sequences for key gen.
- **c/s write MAC secret**
  secret key used to generate MACs by the client (resp. the server)
- **c/s write key secret**
  secret key used by the client (resp. the server) to encrypt application data
- **sequence numbers**
  keep track of received/sent messages
- **initialization vectors**
  for crypto algorithms

Une session peut contenir plusieurs connexions !

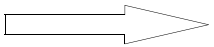
SSL/TLS *Handshake Protocol* Simplifié

Client

Client Hello

- Protocol Version
- Client Random number
- Session ID
- Accepted crypto schemes
- Accepted compression schemes

1



Serveur

Server Hello

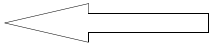
- Protocol Version
- Server Random number
- Session ID
- Selected crypto schemes
- Selected compression schemes

Server Certificate (Optional)
(ev. with CA certification path)

Server Key Exchange (Opt)
(i.e. server public key information)

CertificateRequest (Opt)
(ask the client to provide certificate)

2



SSL/TLS *Handshake Protocol* Simplifié (II)

Client **Serveur**

Client Certificate (Opt.)
(ev. with CA certification path)

Client Key Exchange
Generate *pre_master_secret* and send it encrypted with the server public key

Certificate Verify (Opt.)
Explicit Verification of above certificate (one way function including above master secret)

Finish
For testing purposes. First message protected with above negociated stuff

3 ⇒

⇐ **4**

Finish
Idem. than client

>4 ⇔

Protected Application Data **Protected Application Data**

SSL/TLS: Generation de clés

master_secret =
MD5(pre_master_secret + SHA('A' + pre_master_secret +
ClientHello.random + ServerHello.random)) +
MD5(pre_master_secret + SHA('BB' + pre_master_secret +
ClientHello.random + ServerHello.random)) +
MD5(pre_master_secret + SHA('CCC' + pre_master_secret +
ClientHello.random + ServerHello.random));

key_block =
MD5(master_secret + SHA('A' + master_secret +
ServerHello.random +
ClientHello.random)) +
MD5(master_secret + SHA('BB' + master_secret +
ServerHello.random +
ClientHello.random)) +
MD5(master_secret + SHA('CCC' + master_secret +
ServerHello.random +
ClientHello.random)) + [...];

until enough output has been generated. Then the key_block is partitioned as follows:

client_write_MAC_secret[CipherSpec.hash_size]
server_write_MAC_secret[CipherSpec.hash_size]
client_write_key[CipherSpec.key_material]
server_write_key[CipherSpec.key_material]
client_write_IV[CipherSpec.IV_size] /* non-export ciphers */
server_write_IV[CipherSpec.IV_size] /* non-export ciphers */

SSL/TLS: Remarques Finales

- Les clés secrètes sont le résultat de l'application de fonctions de hachage (MD5,SHA) sur les *random numbers* des enregistrement *Hello* et le *pre_master_secret*
- TLS/SSL est devenu le standard *de facto* pour la sécurité sur le web (à la base de *https*)
- Les clients SSL (*Explorer, Firefox, Opera*, *Chrome*, etc.) contiennent "hard-coded" des certificats correspondant à quelques entités de certification racine (*Verisign*, *Thawte*, *Microsoft*, *RSA*, etc.) permettant de vérifier les certificats présentés par certains serveurs mais SSL est conçu pour s'appuyer sur un réseau global de certification pour le moment inexistant.
- Les failles de sécurité les plus courantes de SSL concernent la génération aléatoire des clés ainsi que les défauts d'implantation les plus courants: *buffer overflows, sql injection*, etc. La faiblesse des fonctions de hachage (MD5, SHA) est aussi un facteur à risque.
- En Novembre 2009[1], on a découvert une attaque permettant à un *Man in The Middle* d'injecter du contenu (*chosen plaintext*) dans un flot authentique suite à une re-négociation des paramètres prévue dans le protocole. Il s'agit d'une **faille dans le protocole** qui a nécessité un patch dans toutes les implantation.
- La faille *heartbleed* basée sur un buffer overflow a serieusement troublé la communauté Internet lors de sa découverte en Avril 2014.

1. Marsh Ray, Steve Dispensa. *Renegotiating TLS. http://extendedsubset.com/Renegotiating_TLS.pdf. Novembre 2009.*

Key Establishment Protocols: Remarques Finales

- Les protocoles d'établissement de clés constituent une pierre angulaire de toute solution de sécurité. Avant de choisir (concevoir) un KEP, il est, donc, indispensable de:
  - Définir les objectifs (confidentialité, authentification d'entités/données, non-répudiation, etc.)
  - Définir le niveau de sécurité souhaité en fonction des propriétés étudiées (*key confirmation*, *perfect forward secrecy*, etc.)
  - Etablir une liste des contraintes liées à l'environnement (utilisateurs, machines, réseau, attaquants potentiels, etc.)
- En fonction de ces critères nous pouvons:
  - Choisir une solution prouvée et robuste (mieux qu'en inventer une *from scratch !*).
  - Vérifier que les objectifs sont atteints et les propriétés satisfaites.
- La vérification des protocoles est un processus complexe et délicat, de plus, les solutions publiées ne sont pas toujours correctes. Deux approches sont possibles (et nécessaires):
  - L'analyse pratique. Analyser les failles du protocole "sur papier" et "sur machine" en tenant compte des pièges classiques: contrôle des nbs. aléatoires pour éviter des *reflection attacks*, redondance des quantités encryptées/signées, etc.
  - L'analyse formelle avec des *logiques* spécialement conçues à cet effet (comme la logique BAN[1]

1. Ban89]: Burrows, Abadi, Needham. *A Logic of Authentication.* Proceedings of the Royal Society, Series A, 426, 1871 (December 1989), 233-271. `http://www.research.digital.com/SRC/personal/Martin_Abadi.`

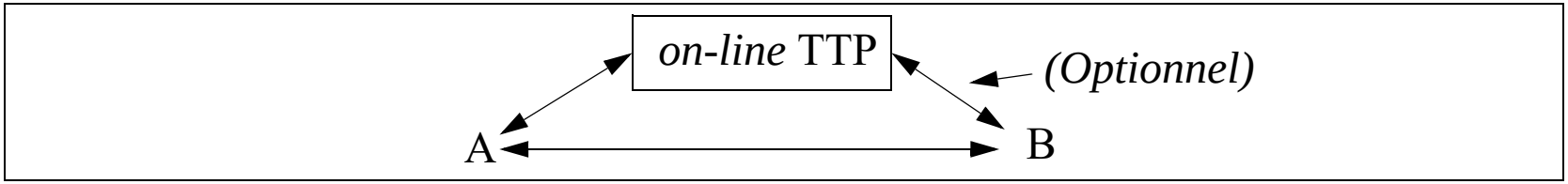
Trusted Third Parties and *Certification*

- *Trusted Third Parties (TTP)*
  - TTP Operational modes
  - KDCs and CAs
  - Other TTPs
- Public Key Authentication
  - Certificates
  - *Certificate Revocation Lists*
  - Authentication trees
  - Certification Topologies
- *Public Key Infrastructures* (PKIs)

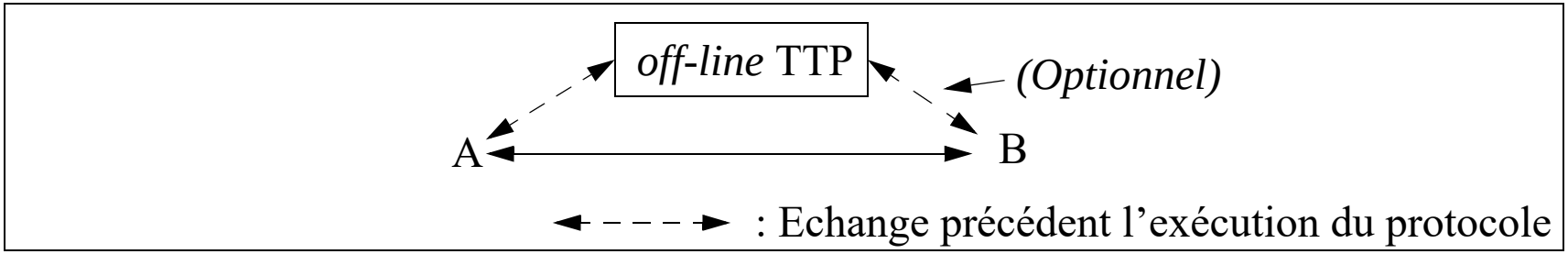
TTP: Modes de Fonctionnement

- *In-line*: Le TTP agit comme intermédiaire pour relayer en temps réel les échanges entre A et B. Exemples: *Proxies*, *Secure Gateways*.

- *On-line*: Le TTP participe en temps réel aux échanges entre A et B mais A et B communiquent directement (sans passer par le TTP). Exemple: *Key Distribution Center.*



- *Off-line*: Le TTP ne participe pas à l'échange en temps réel mais rend l'information disponible *à priori.* Exemple: *Certification Authorities.*



- Comparaison *In-line/On-Line/Off-line*: Echanges facilités et pas besoin de disponibilité permanente des TTP dans le *off-line* (contrairement aux deux autres) mais révocation des privilèges (p.ex.: lorsqu'une clé secrète est compromise) plus complexe.

TTP: *Key Distribution Centers (KDCs)*

- *Key Distribution Centers*: But: Résoudre le n^2 *key distribution problem:*
  - Dans un environnement symétrique de n entités sans intermédiaire:

    `n(n-1)/2 ~ n`^{2} clés différentes sont nécessaires pour toutes les paires d'entités partagent une clé différente.
  - De plus, un tel système n'est pas évolutif (*scalable*) car l'adjonction d'une entité se traduit par la génération de `n` nouvelles clés.

  Si chaque entité partage une clé avec un KDC, seules `n` clés sont nécessaires pour le fonctionnement du système et une clé suffit pour chaque nouvelle entité. L'établissement de canaux sûrs étant assuré par la génération de clés de session et la présence des *tickets à la Kerberos.*
- Problèmes:
  - *Single point of security failure*: par construction le KDC peut usurper l'identité de tous les noeuds du réseau. S'il est compromis tout le système devient vulnérable.
  - *Single point of operational failure*: le mode de fonctionnement habituel d'un KDC est *on-line* (ev. *in-line*). S'il devient indisponible (p.ex. suite à un *denial of service attack*), tout le système est paralysé.
  - *Performance bottleneck*: les opérations des KDC sont souvent coûteuses en temps de calcul (cryptage/décryptage, *random generation*, etc.). Des solutions classiques (ie. le *mirroring*) doivent être envisagées pour répartir la charge des KDCs.

TTP: *Certification Authorities* (*CAs*)

- Le rôle premier d'une Entité de Certification (*Certification Authority* ou *CA*) est d'authentifier l'association entre une entité et sa clé publique (pensez aux attaques *Man-In-the-Middle* !).
- La CA va créer et signer des *certificats* contenant cette association (moyennant une preuve d'identité comme un passeport) et les rendre accessibles aux entités concernées.
- Une fois signées, des copies des certificats (*cached certificates*) peuvent être gardées dans des endroits non protégés (p.ex. dans l'espace disque de l'utilisateur). Cependant, afin de vérifier la signature des certificats, les entités concernées nécessitent une copie *authentique* de la clé publique de la CA.
- *Simpler then safer*: pas besoin d'implanter des protocoles complexes dans une CA.
- Le mode de fonctionnement habituel d'une CA est *off-line*, ce qui diminue les conséquences des périodes (courtes...) d'indisponibilité.
- Problème associé au mode *off-line*: la validité des *cached certificates* peut être remise en question de manière "asynchrone" par un vol de clé privé. Remède: les CAs publient également des listes signées des certificats non valides (*Certificate Revocation Lists* ou CRLs).
- Le compromis d'une CAs a des conséquences *moins évidentes* mais *presque aussi néfastes* que celui d'un KDC surtout si la clé privée servant à signer des certificats est aussi compromise.

CA: *Proof of Possesion* (PoP)

- La vérification de l'identité de A pour créer (ev. modifier) un certificat associant A à sa clé publique n'est pas un critère suffisant. Il faut également vérifier que A possède vraiment la clé privée correspondante.
- Soient A et sa CA: CA_A. Voyons ce qu'un attaquant actif C peut faire en "collaboration" avec une CA_C qui ne vérifierait pas la PoP:

  A signe un document contenant la description d'une invention révolutionnaire et l'envoie à B (le notaire) avec son certificat signé par CA_A:

  A → B: S_{privA}(Invention), S_{privCA_A}(CertA)

  C intercepte ce paquet, s'adresse à CA_C et lui demande de créer un certificat associant son identité C à la clé publique de A et envoie à B:

  C → B: S_{privA}(Invention), S_{privCA_C}(CertC)

  C dévient ainsi l'inventeur révolutionnaire...
- Protocole simple de vérification de PoP:
  CA → A: A, r ; r: nb. aléatoire, A pour protéger A des *chosen message attacks*.
  CA ← A: S_{privA}(A,r); CA n'a plus qu'à vérifier la signature avec `pubA`.
- Ce critère et d'autres critères de comportement comme la mise à jour des CRLs ou la sécurité de la clé de signature introduisent des *niveaux des confiance* pour les CAs et pour les certificats qu'elles signent.
- Ce phénomène s'aggrave avec la prolifération non contrôlée des CAs !

CA: Certification et Révocation

- Problème: Si la même clé sert à signer les certificats et les CRLs, un adversaire possédant la clé privée de signature d'une CA peut attaquer une "victime" A sous l'autorité de cette CA comme suit:
  - Publier une CRL contenant le certificat révoqué de A.
  - Créer un certificat associant A à une clé publique dont il contrôle la clé privée pour ensuite:
  - jouer le *Man-In-the-Middle* pour décrypter les transactions confidentielles pour A;
  - se faire passer par A pour des transactions authentifiées ou des documents signés.
- Solution: *Separation of duties*[1]: La certification et la révocation deviennent des tâches clairement différenciées:
  - Certificats et CRLs sont signés avec des clés différentes,
  - par des entités fonctionnelles différentes (*Certification Authority* et *Revocation Authority*);
  - si possible, résidant dans des machines différentes soumises à des critères de sécurité (*security policies*) indépendants.

1. [Cri96]: Crispo, B. et Lomas, M. *A Certification Scheme for Electronic Commerce*. International Workshop on Security Protocols. Cambridge, UK. April, 1996.

Entités Fonctionnelles Liées à la Certification

- *Name Server*: responsable de la gestion d'un espace de noms unique et cohérent. Lorsque l'authentification est nécessaire, la gestion des noms doit être complétée par la certification des clés publiques associées à ces noms.
  Exemple d'une solution pilote combinant les deux concepts: `DNSsec`: environnement de gestion de noms authentifiés pour Internet.
- *Registration Authority:* Entité chargée d'accomplir les tâches relatives à la gestion des certificats nécessitant un contact direct avec les entités concernées. Ces tâches comprennent la vérification des paramètres nécessaires à la demande initiale ou à la modification des certificats (vérification d'identité, PoP, etc.). Le fait de détacher cette fonctionnalité de la CA est normalement dû à des considérations géographiques.
- *Key Generator*: Permet de déléguer le processus de création de paires de clés publique/ privée à une entité dédiée:
  - Avantages: simplicité pour les utilisateurs; possibilité de renforcer la sécurité des paires choisies.
  - Désavantage: Clé privée connue d'une autre entité! Perte de la non-répudiation.
- *Certificate Directory*: Le répertoire permettant aux utilisateurs d'accéder (en lecture seulement) aux certificats des correspondants.

Autres TTPs

- *Timestamp agent (TA)*: Certifie l'existence d'un document ou le déroulement d'une transaction à un moment bien spécifié dans le temps. Pour ce faire le TA peut:
  - associer un *timestamp* au document (ou à `h(doc)` avec `h` une *Collision Resistant Hash Fonction*) et signer le tout avec sa clé privée et
  - utiliser un *authentication tree* (arbre d'authentification, cf. page 232).
- *Notary agent*: Certifie non seulement l'existence d'un document à un temps donné (comme le TA) mais également sa validité, origine ou appartenance à une entité donnée. Ce service constitue un support (légalement nécessaire ?) pour la non-répudiation.
- *Key escrow agent (KEA)*: Entité autorisée à accéder aux clés secrètes de session pourvu que certaines conditions (p.ex. un mandat judiciaire) soient remplies. Ceci nécessite un système de cryptage dédié. Exemple, le *Clipper key escrow system*:
  - Annoncé en Avril 1993 par l'administration USA, au milieu d'une grande polémique, comme *la* solution de cryptage de communications à grande échelle.
  - Le *Clipper chip* est un dispositif de cryptage/décryptage symétrique donnant accès aux clés des session lorsque les clés secrètes de deux KEAs (normalement des agences fédérales) lui sont fournies en entrée.
  - La présence de quelques failles ainsi que le besoin de crypto asymétrique ont donné lieu à son successeur: le *Capstone chip* pouvant être intégré dans une carte PCMCIA (appelée *Fortezza* et utilisée pour *military level security*).

Authenticité des clés publiques: Certificats

- Un certificat est une pièce d'information associant une entité à sa clé publique. De manière générique, il est constitué des élément suivants:
  - *Serial Number, Version*.
  - *Issuer*: l'identité (global et unique) de la CA signataire.
  - *Signature Algorithm*: l'algorithme permettant de calculer la signature sur le certificat. P.ex.: *MD5 + ElGamal* ou *SHA + RSA*.
  - *Subject*: Le nom (global et unique) de l'entité dont la clé publique est certifiée.
  - *Subject Public Key*: La clé publique de l'entité. Par exemple:
    - **(n,e)**: modulus et exposant publique pour RSA.
    - **(p,** α, $\alpha^{\mathbf{x}}$ **mod p)**: modulus, générateur et partie publique pour Diffie-Hellman.
  - *Subject Public Key Algorithm*: L'algorithme associé à la clé publique. P.ex: RSA ou Diffie-Hellman.
  - *Validity*: La période de validité du certificat, normalement exprimée en UTC.
  - *Signature*: Contient la signature effectuée au moyen du *Signature Algorithm* et de la clé privée de la CA. Elle porte sur l'ensemble des enregistrements précédents et garantit ainsi l'authenticité des informations qu'ils contiennent.

Certificate Revocation Lists (CRLs)

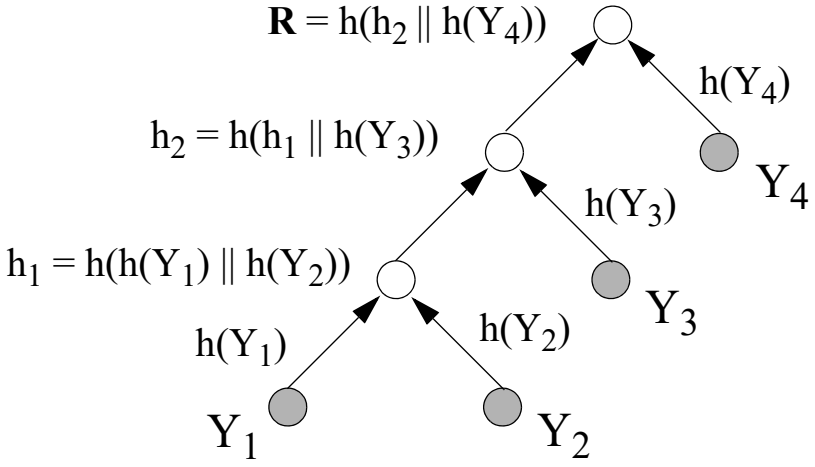
- Il s'agit de listes contenant des certificats devenus non valables suite à une clé privée compromise ou à tout autre facteur mettant en évidence la validité des informations contenues dans un certificat (changement de l'algorithme utilisé, changement de fonction pour un *role-based certificate*, etc.).
- Une CRL générique a les éléments suivants:
  - *Issuer*, *Signature Algorithm*: comme pour les certificats.
  - *Date of Issue*, *Date of Next Issue*: date d'émission et date de la prochaine émission.
  - Pour chaque certificat révoqué, les enregistrements suivants:
    - *Serial Number* du certificat révoqué.
    - *Revocation Date*.
  - *Signature*: signature portant sur toute la liste.
- Une CA se doit de publier des CRLs avec une fréquence très élevée et en utilisant des canaux de distribution de large audience, afin de diminuer le risque de fraudes.
- La révocation est le *talon d'Achilles* de tout système à clés publiques...
- Une solution: certificats avec des lapses de validité très courts (quelques minutes) exigeant une re-confirmation périodique de la part des CAs...
- ...mais ceci nous fait revenir au mode *on-line* et à imposer, donc, une grande disponibilité de la part des CAs.

Arbres d'Authentification

- Les arbres d'authentification sont une alternative à la certification pour authentifier des information publiques.
- Il s'agit d'exploiter les avantages d'une structure d'arbre (normalement binaire) avec l'utilisation de *hash fonctions* et l'authentification du noeud *racine*.

Soit un arbre A avec n feuilles. Soit h une *collision resistant hash function* (CRHF). L'arbre A peut être utilisé pour l'authentification de n valeurs publiques Y_1, Y_2, ...,Y_n en construisant un arbre d'authentification comme suit:

1) Les valeurs Y_1,Y_2,..., Y_n sont placées dans les feuilles de l'arbre.
2) Chaque arc partant d'une feuille Y_i est étiqueté $h(Y_i)$ (h étant une CRHF).
3) Chaque noeud non-terminal ayant des arcs sousjacents étiquetés h_1 et h_2 est étiqueté $h(h_1 || h_2)$ (|| dénote concaténation).

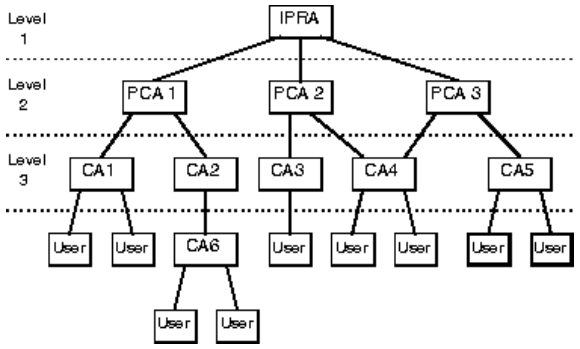


Arbres d'Authentification (II)

- Pour vérifier l'authenticité de Y_1, il est nécessaire de fournir les valeurs $h(Y_2)$, $h(Y_3)$, $h(Y_4)$. Après, il suffit de calculer $h(Y_1)$, h_1 et h_2 (selon la figure) et accepter l'authenticité de Y_1 si $h(h_2 \,||\, h(Y_4)) = \mathbf{R}$. Une modification illicite dans Y_1 se traduirait (par les caractéristiques de la CRHF) en une valeur différente pour $h(h_2 \,||\, h(Y_4)) <> R$.
- A noter que seule la valeur **R** doit être authentifiée (p.ex. à l'aide d'une signature digitale). Les autres valeurs sont protégées par la *non-réversibilité* de la CRHF.
- Avantage: Seul R nécessite une protection cryptographique pour l'authentification!
- Inconvénients:
  - Pour vérifier la valeur Y_1, les valeurs $h(Y_{2,3,4})$ et la valeur R sont nécessaires. Pour minimiser cet effet, on peut d'utiliser des arbres *équilibrés* (des arbres dont les chemins différent d'*au plus* un arc) afin de réduire le nombre de données intermédiaires à $\sim \log_2 n$.
  - Lorsqu'un noeud est modifié, tout le chemin jusqu'à la racine doit être re-calculé.
  - Lorsque des nouveaux noeuds sont rajoutés, il convient de construire des arbres non-équilibrés (comme celui de la figure) et de rajouter les noeuds par la racine.
- Application principale: *timestamping*: le *timestamping agent* (TA) construit un tel arbre et fournit au requérant le *timestamp* signé avec sa clé privée ainsi que le chemin de vérification. TA publie **R** quotidiennement dans un journal ce qui lui empêche de tricher!

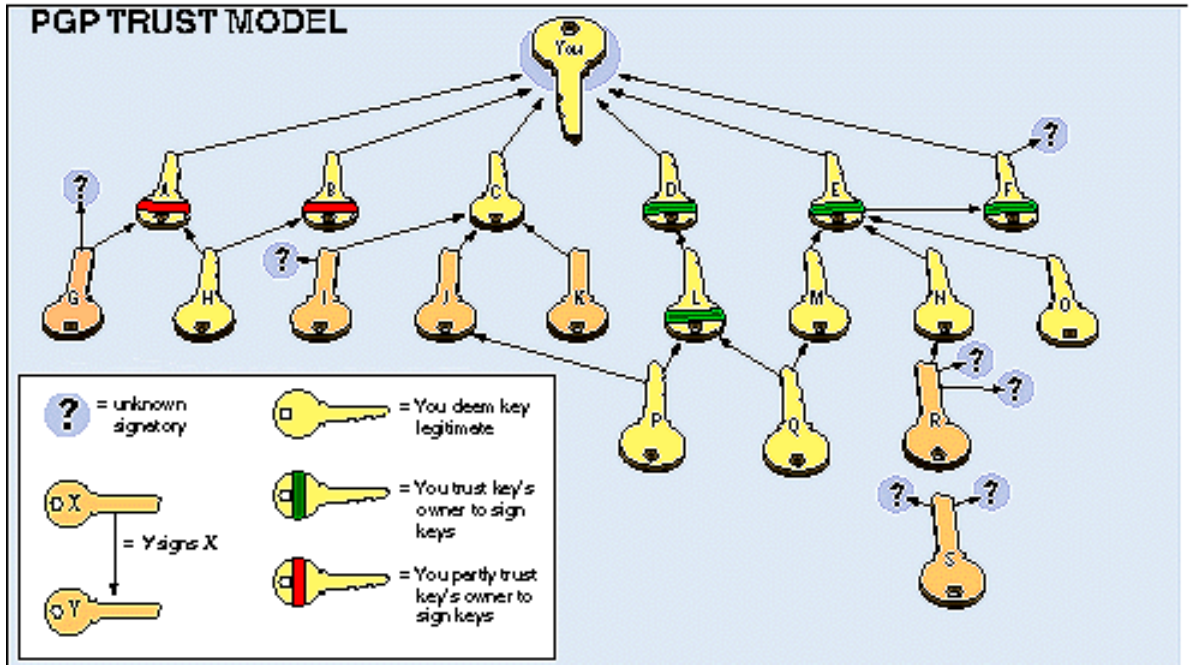
Topologies de Certification

- Lorsque deux utilisateurs appartenant à des CAs différentes souhaitent communiquer, il apparaît un problème de confiance: doit-on faire confiance à un certificat émis par une autre CA?.
- Le processus de *certification* croisée (*cross-certification*) permet à CA_A de certifier la clé publique pub_{CAB} de CA_B. Le certificat résultant s'appelle certificat croisé (*cross-certificate*), on le note: $CA_A\{CA_B\}$.
- Si A désire vérifier l'authenticité de la clé publique de B et il existe un certificat croisé $CA_A\{CA_B\}$, A va demander à B de lui fournir son certificat signé par CA_B, soit $CA_B\{B\}$. La chaîne de certification résultante: $CA_A\{CA_B\}CA_B\{B\}$ permet à A de vérifier la clé publique de B en utilisant une copie authentique de pub_{CAA}.
- La relation de confiance nécessaire à la certification croisée n'est pas toujours facile à établir dans des environnements concurrents, c'est pourquoi des modèles hiérarchiques entre les CAs ont été proposés. Exemple le modèle hiérarchique strict de PEM/X.509:



Topologies de Certification (II)

- Dans l'environnement PEM , toute chaîne de certification non-locale commence au noeud racine, dont la clé publique est supposée connue du monde entier...
- D'autres modèles comme celui proposé par PGP se basent sur une structure de graphe où les noeuds sont les utilisateurs qui agissent comme CAs pour certifier les clés publiques des correspondants. Même si bien adapté pour des groupes fermés d'utilisateurs, ce modèle a ses limites lorsqu'il est appliqué à des populations non connectées.



- D'autres schémas proposés combinent la structure hiérarchique avec la certification croisée bidirectionnelle.
- Il faut garder les chaînes de certification aussi courtes que possible (une chaîne est toujours aussi vulnérable que son maillon le plus faible!).

Public Key Infrastructure (PKI): Définitions

Définition: *Une PKI est une infrastructure intégrée permettant de fournir un ensemble de services de sécurité sur la base de la cryptographie à clés publiques.*

Entités Fonctionnelles:

- **Entité de certification** (*Certification Authority* ou CA): Entité responsable de la création et maintenance des certificats.
- **Répertoire des certificats** (*Certificate Repository*) mettant les certificats à disposition des utilisateurs et des applications. Technologies utilisées: X.500, LDAP, Serveurs WWW, DNS, etc.
- **Révocation des certificats** (*Certificate Revocation*) compromis ou devenus obsolètes (notamment gestion des CRLs)
- **Sauvegarde et rétablissement centralisés des clés** (*Key Backup and Recovery*): Entité permettant de gérer la perte de clés suite à des événements divers: destruction du support matériel , oubli du mot de passe de déblocage, départ de l'employé, etc. A noter que cette procédure s'applique principalement à la *clé privée de décryption* (par opposition à la *clé privée de signature*).
- **Mise à jour automatique des clés** (*Automatic Key Update*) après la fin de leur validité.

Public Key Infrastructure (PKI): Entités Fonctionnelles (II)

- **Historique des clés et des certificats** (*Key and Certificate History*). Cette entité permet de récupérer des clés devenues obsolètes, ayant servi à encrypter un document dans le passé.
- **Certification croisée** (*Cross-Certification*) avec d'autres PKI (clients, fournisseurs, partenaires, etc.). Cette fonctionnalité permet (sous certaines contraintes) de valider les certificats émis par d'autres PKIs
- **Support pour la *non-répudiation***: Service à valeur ajouté permettant de fournir l'évidence nécessaire à démontrer le déroulement d'une transaction authentifiée (*data origin authentication*, *time-stamped data signature*, *signed receipt of delivery*, etc.)
- ***Secure Time Stamping***: Entité capable de fournir un temps de référence accepté par tous les intervenants d'une PKI. Applications principales: *non-répudiation*, arbitrage en cas de conflits, etc.
- **Logiciel Client**: Cette entité fonctionnelle permet de réaliser toutes les opérations propres à la PKI côté client. Exemples: gestion des certificats utilisateurs, signature de documents, décryption d'information, gestion de périphériques spécifiques (lecteurs de cartes à puces, dispositifs biométriques, etc.)

PKI: Principaux Avantages et Inconvénients

Avantages

- **Sécurité**: La nature intégrée d'une PKI permet de créer un environnement de sécurité sans maillons faibles.
- **Tout en un**: Une PKI permet l'intégration et la gestion de tous les paramètres de sécurité propres à un grand nombre de services: authentification forte d'entités, signature des documents permettant la non-répudiation, *single sign-on*, réseaux privés virtuels (VPNs), communications sécurisées avec des clients/partenaires/fournisseurs (B2C,B2B), etc. La PKI constitue une économie notable par rapport aux solutions "au cas par cas".
- **Inter-opérabilité *intra* et *inter* entreprise**: Les principaux produits PKI répondent à des normes de standardisation très répandues (X.509,PKCS,OCSP,etc.). Un grand nombre d'applications et dispositifs matériels sont désormais conformes à ces standards. La compatibilité possible entre différents fournisseurs de PKIs permet également (sous quelques réserves) l'inter-opérabilité inter-entreprise.

Inconvénients

- **Coût de mise en place:** produits chers, compétences rares
- **Complexité**...mais:
- la **sous-traitance** du "service" PKI est une alternative.