Cours 4MMCSR - Codage et sécurité des réseaux

- Objectif: présentation des aspects principaux à prendre en compte pour construire un système informatique distribué fonctionnant de manière sûre, même en contexte ouvert (i.e. en présence d'erreurs aléatoires et/ou d'attaques malicieuses).

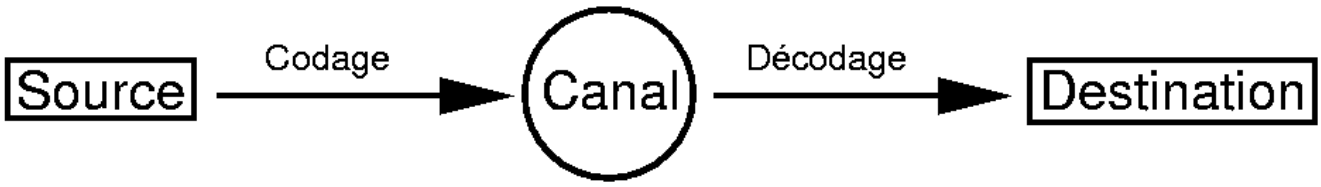
- L'accent est mis sur:
  - les technologies appliquées de cryptologie et de codage permettant de garantir l'intégrité des communications, et leur intégration effective illustrée sur des applications de télécommunications;
  - la sécurité appliquée: les diverses attaques à considérer, car pour pouvoir effectuer de la sécurité défensive, il est nécessaire de savoir attaquer. La lecture et la compréhension d'articles de recherche récents relatifs à la sécurité appliquée seront abordées.

Plan du cours

- **Partie 1. Technologies de codage numérique et intégrité des communications**
  [C Lauradoux, JL Roch]
  - Introduction: technologies de base en cryptologie: confidentialité (chiffrement symétrique et asymétrique), authentification (challenge, signature), intégrité (hachage)
  - 2. Codage :
    - Détection d'erreurs dans les réseaux – Codeurs et décodeurs CRC (circuits LFSR) . Exemples (Ethernet et GSM).
    - Codes correcteurs d'erreurs par interpolation (Reed-Solomon). Application.
    - Rafales d'erreurs et entrelacement. Codes CIRC pour les CD; disques RAID.
  - 3. Cryptologie :
    - chiffrement symétrique et Chiffrement asymétrique; (ECDLP/El Gamal). Fonctions de hachage et générateurs aléatoires.
    - Application aux attaques par corrélation. Exemple: Siegenthaler sur GSM.
- **Partie 2 : Sécurité applicative et attaques** **[F Duchene, K Hossen]**
  - 1. Sécurité des applications Web et des réseaux.
  - 2. Partage de clefs et architectures PKI.
  - 3. Overflows et Shellcode
  - 4. Fuzzing de protocoles
  - 5. Recherche: avancées en test de la sécurité de protocoles

INTRODUCTION

Notion de code



- Le code doit répondre à différents critères :
  - Rentabilité : compression des données.
  - Sécurité de l'information : cryptage, authentification, etc.
  - Tolérance aux fautes : correction/détection d'erreurs.

Théorie des codes : théorèmes de Shannon 1948

- Compression : « Pour toute source X d'entropie H(x) on peut trouver un code dont la longueur moyenne s'approche de H(X) d'aussi prêt que l'on veut »
  - Algorithme d'Huffman, extensions de sources
- Correction d'erreurs : « Pour tout canal on peut toujours trouver une famille de codes dont la probabilité d'erreur après décodage tend vers 0 »
- Cryptage : « si un chiffrement est parfait, alors il y a au moins autant de clefs possibles que de messages »
  - Un cryptanalyste doit obtenir au moins H(M) informations pour retrouver M

À quoi sert la cryptographie (CAIN) ?

- Confidentialité des informations stockées ou manipulées
  - Seuls les utilisateurs autorisés peuvent accéder à l'information
- Authentification des utilisateurs
  - L'utilisateur est-il ou non autorisé ? Pour quelle action ?
- Intégrité des informations stockées ou manipulées
  - Contre l'altération des données
- Non-répudiation des informations
  - Empêcher un utilisateur de se dédire

Plan du cours

- Lien entre Information et secret
  - Chiffrement parfait
  - cryptographie symétrique
  - fonction de hachage cryotographique, intégrité
  - cryptographie asymétrique, signature
- Intégrité et correction d'erreurs.
  - Codes détecteurs d'erreurs, CRC
  - Codes linéaires
  - Codes de Reed-Solomon (cycliques)

- *NB : utilisation de connaissances de base (non détaillées)*
  - *Théorie de l'Information*
  - *Algèbre "discrète" et arithmétique (corps finis)* [Z/pZ, *un peu* GF(2^m)]

Cryptographie

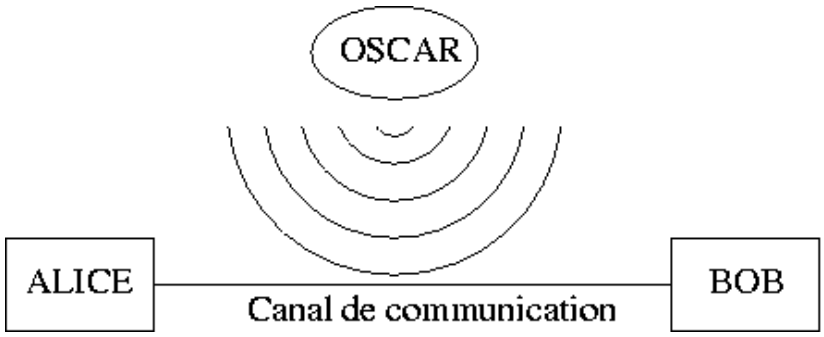
- Information chiffrée

  Connaissance de l'existence de l'information

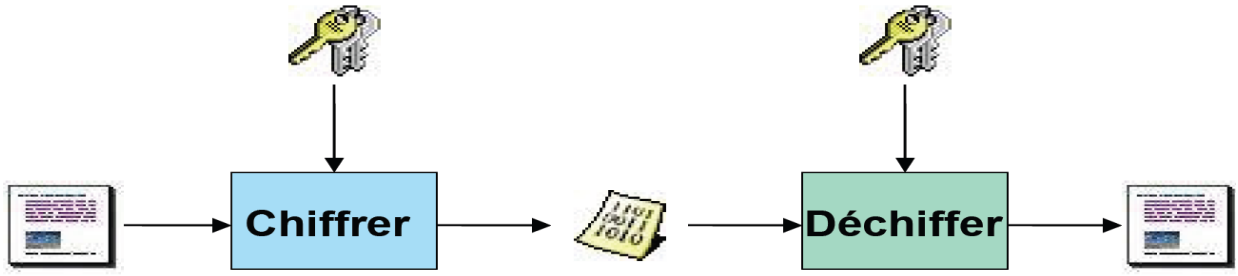
  ≠

  Connaissance de l'information

- Objectif
  - Permettre à Alice et Bob de communiquer sur un canal peu sûr
    - Réseau informatique, téléphonique, etc.
  - Oscar ne doit pas comprendre ce qui est échangé



Algorithmes de cryptographie



- Propriétés théoriques nécessaires :

1. Confusion

Aucune propriété statistique ne peut être déduite du message chiffré

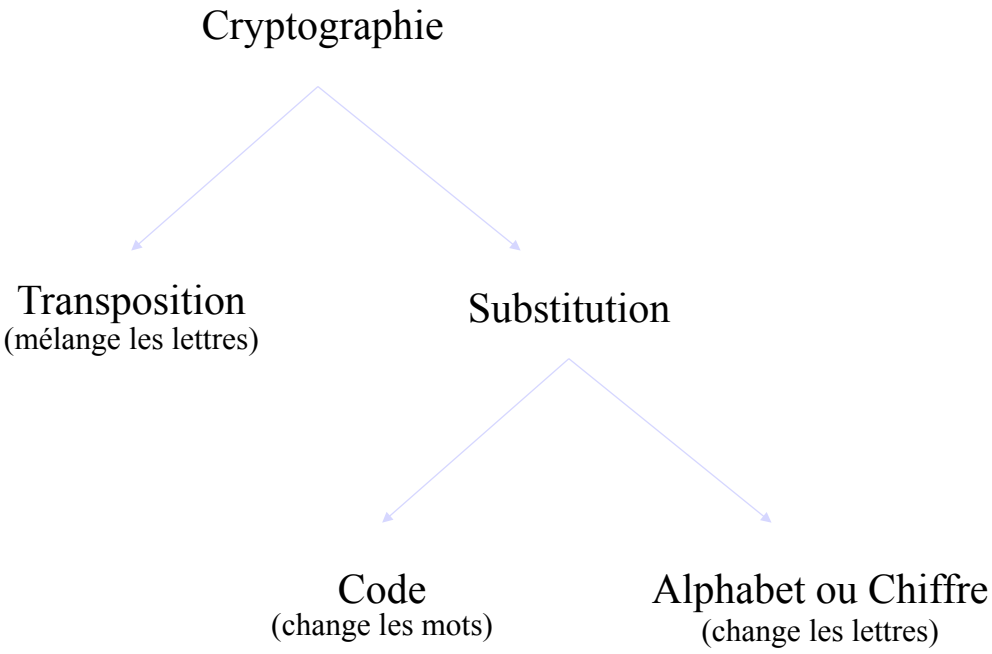
2. Diffusion

Toute modification du message en clair se traduit par une modification complète du chiffré

Terminologie

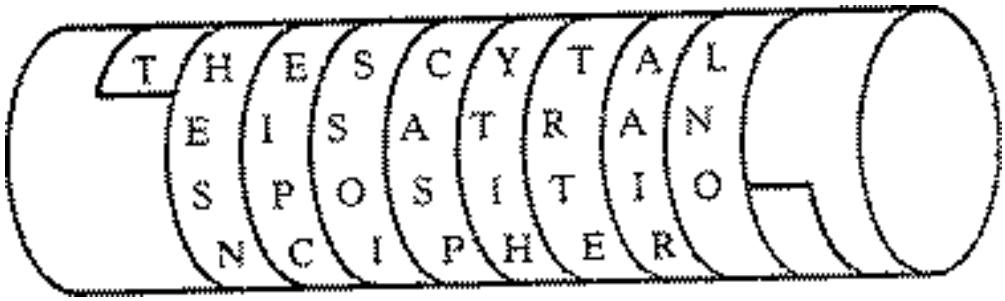
- Texte clair
  - information qu'Alice souhaite transmettre à Bob
- Chiffrement
  - processus de transformation d'un message M de telle manière à le rendre incompréhensible
    - Fonction de chiffrement E
    - Génération d'un chiffre (message chiffré) C = E(M)
- Déchiffrement
  - processus de reconstruction du message clair à partir du message chiffré
    - Fonction de déchiffrement D
    - D(C) = D(E(M)) = M (E est injective et D surjective)

Cryptographie *ancienne*



Transposition

- Chiffrement de type anagramme : mélange les lettres du message
- Sécurité théorique
  - Message de 35 lettres : 35! chiffrés possibles
- Problèmes
  - Confusion sur la syntaxe mais … chaque lettre conserve sa valeur
  - Clé de chiffrement « complexe »
  - Ex: Scytale spartiate (5ème siècle av JC)



Substitution

- Chiffrement en changeant d'alphabet
  - Kama sutra : mlecchita-vikalpà (art de l'écriture secrète, 4ème siècle av JC)
- Sécurité théorique
  - Alphabet de 26 lettres : 26! alphabets possibles
- Problèmes
  - Confusion sur l'alphabet mais … chaque lettre conserve sa place d'origine
  - Ex: Chiffrement de Jules César (1er siècle av JC)

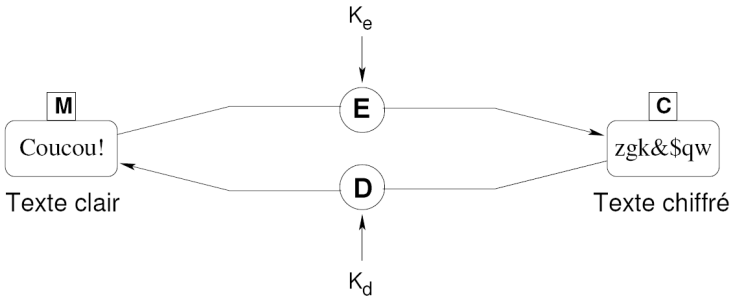
Alphabet clair : abcdefghijklmnopqrstuvwxyz
Alphabet chiffré : DEFGHIJKLMNOPQRSTUVWXYZABC

Texte clair : errare humanum est, perseverare diabolicum
Texte chiffré : HUUDUH KXPDQXP HVW, SHUVHYHUDUH GLDEROLFXP

Cryptographie moderne

- Principes de Auguste Kerckhoffs (1883)
  1. La sécurité repose sur le secret de la clef et non sur le secret de l'algorithme
  – Canal +, Cartes Bleues, PS3 !!!
  2. Le déchiffrement sans la clef doit être impossible (à l'échelle humaine)
  3. Trouver la clef à partir du clair et du chiffré est impossible (à l'échelle humaine)

Types de cryptographie

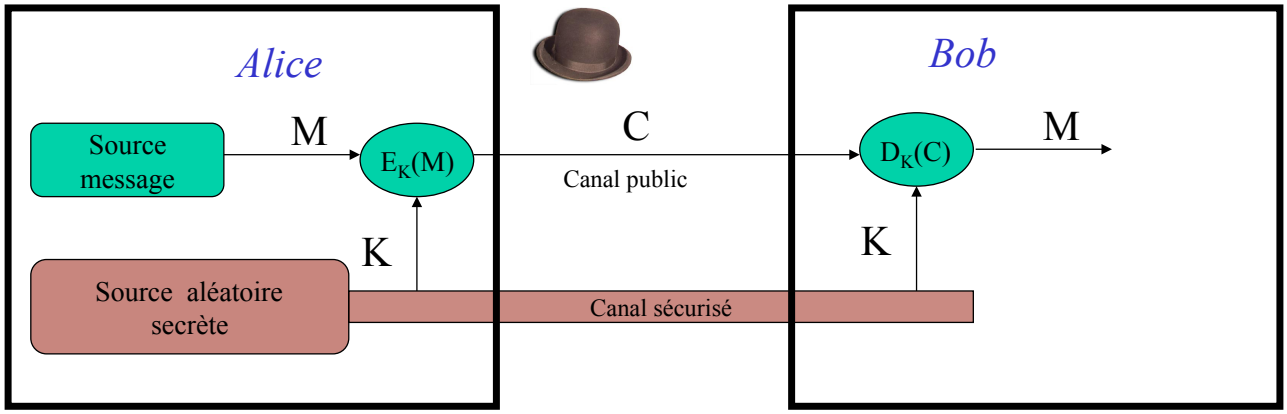


- En pratique E et D sont paramétrées par des clefs K_d et K_e
- Deux grandes catégories de systèmes cryptographiques
  - Systèmes à clefs secrètes (symétriques) : $K_e=K_d=K$
  - Systèmes à clefs publiques (asymétriques) : $K_e \neq K_d$
- Deux types de fonctionnement
  - Par flot : chaque nouveau bit est manipulé directement
  - Par bloc : chaque message est découpé en blocs

Cryptographie symétrique

Chiffrement symétrique

- Définition : chiffrement **parfait** ssi l'attaquant n'a **aucune information** sur M



- Théorème 1 [Shannon49] : si le chiffrement est parfait, alors
  [H = entropie = qté info = incertitude] **H(K) ≥ H(M)**
  - conséquence: la clé secrète doit être de taille supérieure à M (compressé)
- Théorème 2. Le chiffrement de **Vernam** garantit un **chiffrement parfait**

Exemple de chiffrement symétrique : OTP

- **« OTP » : One-time-pad** [AT&T Bell Labs], Vernam (USA 1917)
  - C = M ⊕ K (xor bit à bit avec une clef K secrète
  - Téléphone rouge
- Cryptanalyse possible uniquement si
  - Mot-clef trivial
  - Réutilisation du mot-clef : $(M_1 \oplus K) \oplus (M_2 \oplus K) = M_1 \oplus M_2$!!!
    ⇨ Alors des morceaux de textes en clair donnent $M_1 \oplus M_2 \oplus N$ ¼ M_1
- Premier chiffrement avec preuve de sécurité (cf. entropie)
  - Sous condition que la clef soit d'entropie plus grande que le message et non réutilisée
  - Clef choisie uniformément de même taille que le message

Vernam est un chiffrement parfait

- $|K|=|M|=|C|$, les clefs sont équiprobables, $c = m \oplus k$
- Parfait ssi $P(M=m|C=c) = P(M=m)$

- Or $P(C=c) = \sum P(K=k)\, P(M=D_k(c))$
  $= \sum 1/|K|\, P(M=D_k(c))$ *équidistribution*
  $= 1/|K| \sum P(M=m)$ *$\oplus$ bijectif*
  $= 1/|K|$
- Également $P(C=c|M=m) = P(K=m\oplus c)$ *$\oplus$ bijectif*
  $= P(K=k) = 1/|K|$

- Donc : $P(M=m|C=c) = P(C=c|M=m)P(M=m)/P(C=c)$
  $= P(M=m)$ *[Bayes]*

Chiffrement à clef sécrète: DES et AES

- DES [1972, IBM] Data Encryption Standard
  - Exemple : Crypt unix
  - Principe : chiffrement par bloc de 64 bits
    - clef 64 bits / 56 bits utiles + 16 transformations/rondes
    - Chaînage entre 2 blocs consécutifs
  - Attaque brutale : 2^56 = 64.10^15
    - 1000 PCs * 10^9 Op/s * 64000s = 20h !!!!
    - Double DES (???), Triple DES,...
- En Oct 2000 : nouveau standard AES
  - Advanced Encryption Standard www.nist.gov/AES
  - Corps à 256 éléments : F256

Cryptographie à clef publique (Cryptographie asymétrique)

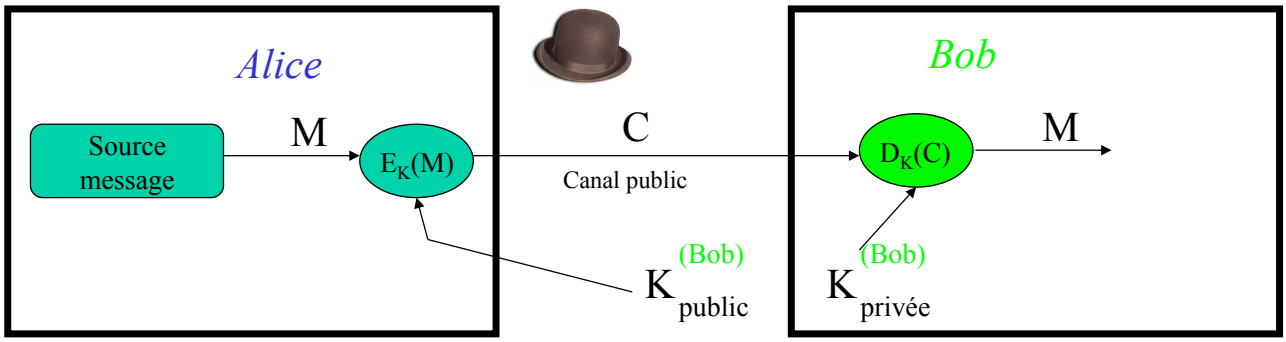
Fonction à sens unique

« clef » publique de Bob

Clef privée de Bob

Chiffrement asymétrique

- Asymétrique = boîte à lettres : C contient toute l'information sur M



- Définition : parfait si, connaissant C et la fonction E, alors il est **calculatoirement impossible** de calculer M
  - Humain: 10^{10}ordis (!!) *10^{15} op/sec (!!) * 128 bits (!) * 3000 ans < 10^{39} = 2^{130} op binaires
  - Terre : 10^{50} opérations =
  - Univers : 10^{125} = 2^{420} opérations « physiques » élémentaires depuis le bing bang
- Les fonctions E et D = E^{-1} doivent vérifier :
  - X = E(M) : facile à calculer : coût (E (M)) = « linéaire » en la taille de M
  - M = D(X) : calculatoirement impossible
- Existe-t-il de telles fonctions ***à sens unique*** ???

Un exemple de chiffrement asymétrique: RSA

Rivest / Shamir / Adleman (1977)

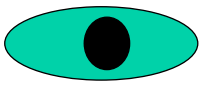
1. Chiffrement RSA : E et D
2. Validité de RSA
   1. E (D (x)) = D(E(x)) = x
   2. E est facile à calculer
   3. E est difficile à inverser sans connaître D
3. Signature RSA
   1. Directe
   2. Avec résumé du message

Chiffrement RSA

Alice

Veut envoyer M secret à Bob

Eve



Bob

1/ **Construction clefs Bob**

- p, q premiers grands
- $n = p \cdot q$
- $\varphi(n) = (p-1).(q-1)$
- e petit, premier avec $\varphi(n)$
- $d = e^{-1} \pmod{\varphi(n)}$

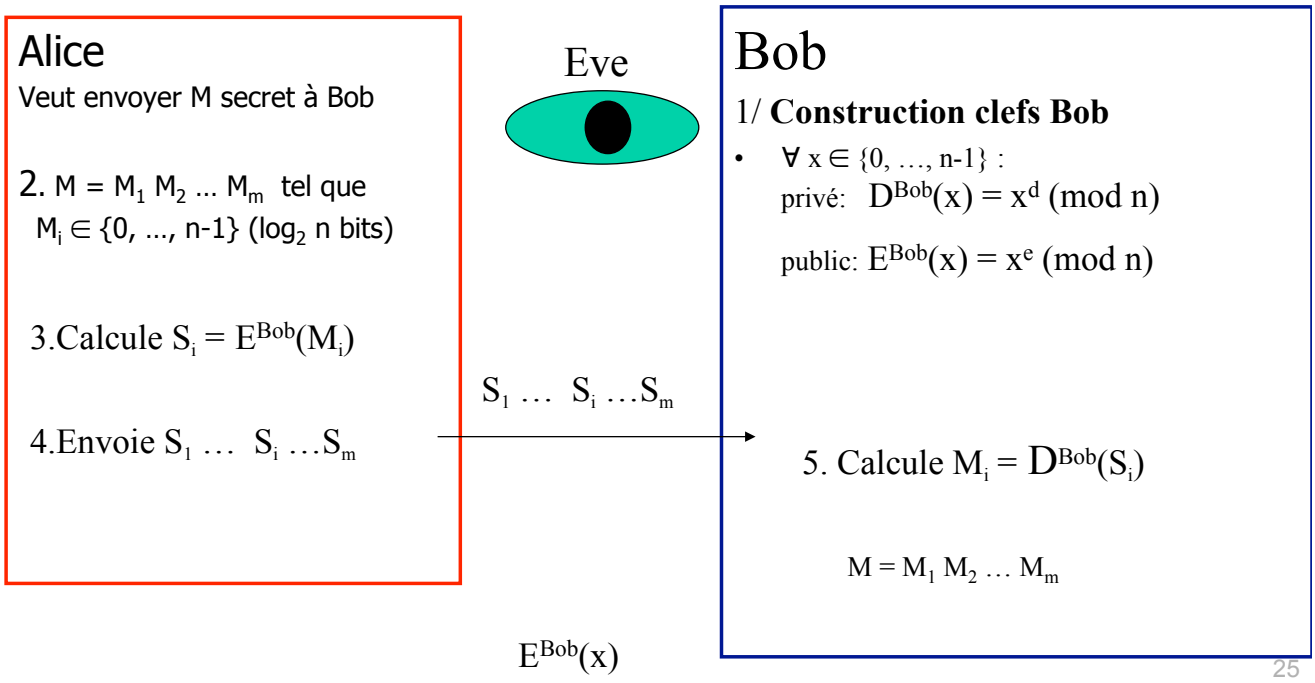
- Clé privée: (d, n)
  Clé publique: (e, n)
- $\forall x \in \{0, \ldots, n-1\}$:

$$D^{Bob}(x) = x^d \pmod{n}$$

$$E^{Bob}(x) = x^e \pmod{n}$$

$E^{Bob}(x)$

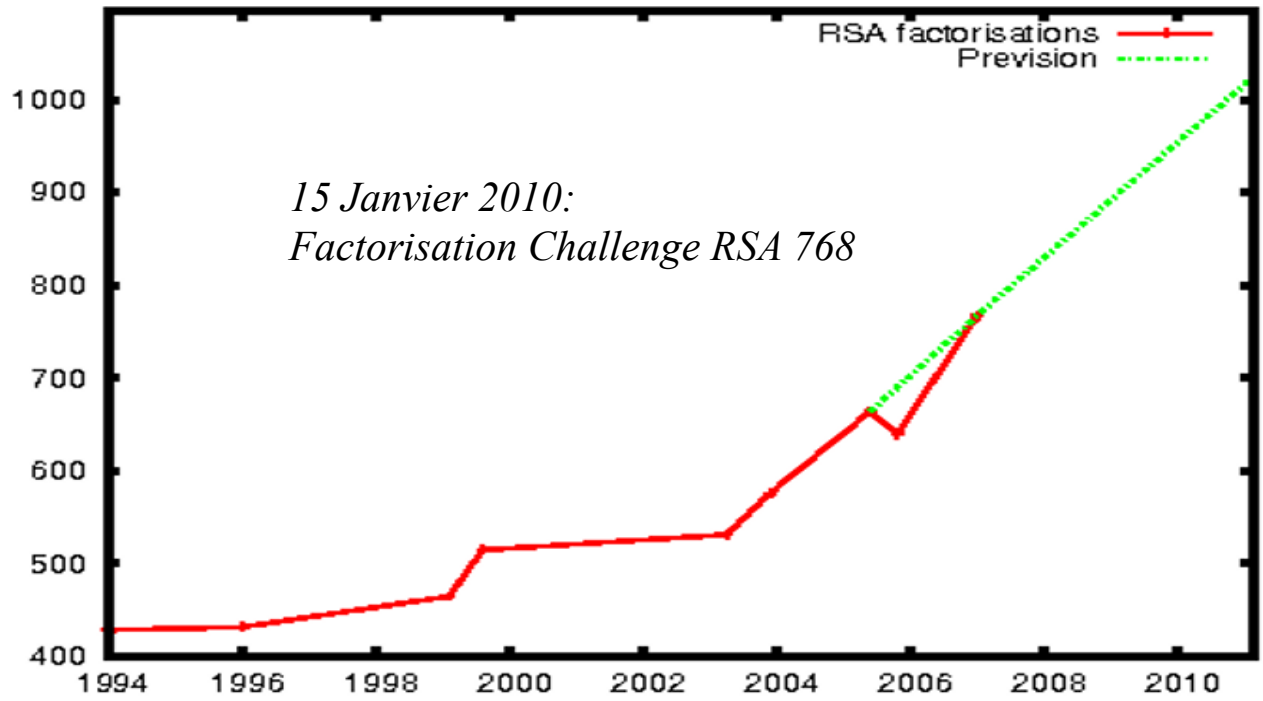
Chiffrement RSA



Validité de RSA

- D^{Bob} est la réciproque de E^{Bob} :
  - $\forall x \in \{0, \ldots, n-1\}$: $D^{Bob}(E^{Bob}(x)) = E^{Bob}(D^{Bob}(x)) = x$
- E^{Bob} est une fonction à sens unique *chausse-trappe*:
  - $E^{Bob}(x)$ est peu coûteuse à calculer
  - $D^{Bob}(x)$ est peu coûteuse à calculer
  - Calculer *d* à partir de (*n* et *e*) est calculatoirement aussi difficile que factoriser *n*
- Générer une clef RSA [(n,d), (n,e)] est peu coûteux

Clefs RSA sur 1024 bits ?



Niveaux d'attaque

- Étude de la sécurité des procédés de chiffrement

1. Texte chiffré connu : seul C est connu d'Oscar
2. Texte clair connu : Oscar a obtenu C et le M correspondant
3. Texte clair choisi : pour tout M, Oscar peut obtenir le C
4. Texte chiffré choisi : pour tout C, Oscar peut obtenir le M

- Garantir la confidentialité
  - Impossible de trouver M à partir de E(M)
  - Impossible de trouver la méthode de déchiffrement D à partir d'une séquence $\{M_1,\ldots,M_k,E(M_1),\ldots,E(M_k)\}$

Algorithmes d'attaque

1. Attaque exhaustive (par force brute)
   - Énumérer toutes les valeurs possibles de clefs
   - 64 bits → 2^{64} clefs = 1.844 10^{19} combinaisons

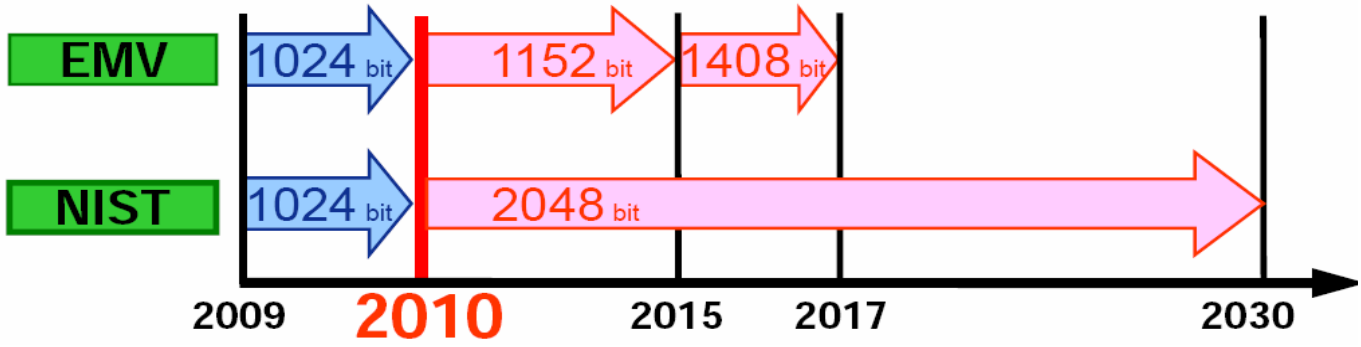
     Un milliard de combinaisons/seconde → 1 an sur 584 machines
2. Attaque par séquences connues
   - Deviner la clef si une partie du message est connu

     ex: en-têtes de standard de courriels
3. Attaque par séquences forcées
   - Faire chiffrer par la victime un bloc dont l'attaquant connaît le contenu, puis on applique l'attaque précédente …
4. Attaque par analyse différentielle
   - Utiliser les faibles différences entre plusieurs messages (ex: logs) pour deviner la clef
- etc

Attaques - quelques chiffres

- La résistance d'un chiffrement dépend du nombre d'opérations requis pour le casser sans connaître le secret
- #opérations effectuées par l'univers depuis le big-bang : 10^{123}
  - Nombre particules dans l'univers : 10^{100}
- Echelle de Borel
  - 10^{10} = échelle humaine : attaque humaine
  - 10^{20} = échelle terrestre : attaque terrestre
  - 10^{100} = échelle cosmique : attaque cosmique
  - $>10^{100}$ = attaque super-cosmique
- Taille de clef et attaque exhaustive :
  - •Clé de 128 bits aléatoire : $2^{128} = 10^{36}$
  - •Clé de 256 bits aléatoire : $2^{256} = 10^{75}$
  - •Clé de 512 bits aléatoire : $2^{512} = 10^{150}$
  - •Mais attention aux failles !!!

Recommandations EMV, NIST



Niveau de Sécurité estimés

bits	80 (SKIPJACK)	112 (3-DES)	128 AES-small	192 AES-medium	256 AES-large
RSA	1024	2048	3072	7168	13312
DLP	1024	2048	3072	7168	13312
EC DLP	192	224	256	384	521

- Factorisation ¼ $Ln[1/3,1.923^{+o(1)}] = \exp((1.923^{+o(1)})\ln(n)^{1/3}\ln(\ln(n))^{2/3})$
- Calcul d'index pour le log discret ¼ $Lp[1/3,1.923^{+o(1)}]$
- DLP sur courbes elliptiques : Pollard rho ¼ $O(n^{1/2})$

Intégrité et répétition

- Alice utilise OTP pour communiquer avec Bob.
  Elle veut envoyer M, et calcul C = M xor $K_{AliceBob}$
- Bob reçoit C', qu'il décode en M' = C' xor $K_{AliceBob}$
- Quelle "confiance" Bob et Alice peuvent-ils dans le fait que M = M' ? (*valeur de la confiance*)
- Comment augmenter cette confiance ?
  - Contrôle d'intégrité
  - Naïf: répétition

Fonction de hachage cryptographique

- **Déf**: Une fonction h: $\{0,1\}^+ \rightarrow \{0,1\}^k$ qui, à tout message M, associe un résumé sur k bits (k fixé, par exemple k=512)
  et telle qu'il est **calculatoirement** impossible *(bien que mathématiquement possible)*:
  - étant donné r , de trouver M tel que h(M) = r [*résistance à la préimage*] ;
  - étant donné M, de trouver M'≠M tel que h(M)=h(M') [*résistance à la 2ème préimage*];
  - trouver M'≠M tel que h(M)=h(M') [***résistance aux collisions***] .
- **Application**: h(M) est une empreinte numérique sur k bits du message M.
  Il est calculatoirement impossible d'avoir 2 messages différents de même "hash" (empreinte, résumé)
  => permet de vérifier l'intégrité d'un message.
- **Exemples**:
  - MD5 (128 bits, attention: non résistante aux collisions), RIPEMD, Whirlpool
  - Standard: SHA = Secure Hash Algorithm
    - 1993: SHA-0, SHA-1 : 160 bits (mais collisions)
    - 2000: SHA-256 et SHA-512 (SHA-2, variante de SHA-1): résumés sur 256 et 512 bits
    - 2012: SHA-3

Fonction de hachage cryptographique: le couteau suisse de la cryptographie

- Nombreuses applications:
  - signatures numériques (avec des algorithmes à clef publique)
  - Générateur de nombres pseudo-aléatoires
  - Mise à jour et dérivation de clef (symétrique)
  - Fonction à sens unique (asymétrique)
  - MAC Message authentication codes (avec une clef secrète)
  - Intégrité
  - Reconnaissance de programmes ou codes binaire
    (lists of the hashes of known good programs or malware)
  - Authentication d'utilisateurs (with a secret key)
  - Non-répudiation (suivi de versions, "commit")

Partie - Correction d'erreurs

- Dans les systèmes électroniques digitaux, l'information est représentée en format binaire : uniquement par des 0 et des 1 (bits)

- Transfert d'information d'un point à un autre : il y a toujours une chance pour qu'un bit soit mal interprété (1 au lieu de 0 et vice versa)
  Cela peut avoir de multiples causes, par exemple :
  - Bruit parasite
  - Défauts au niveau des composants
  - Mauvaise connexion
  - Détérioration due au vieillissement …

- La correction d'erreurs est le procédé utilisé pour :
  - *détecter automatiquement* et *corriger automatiquement* ces bits erronés.

⇒ Au niveau logiciel ou au niveau matériel (pour le haut débit).

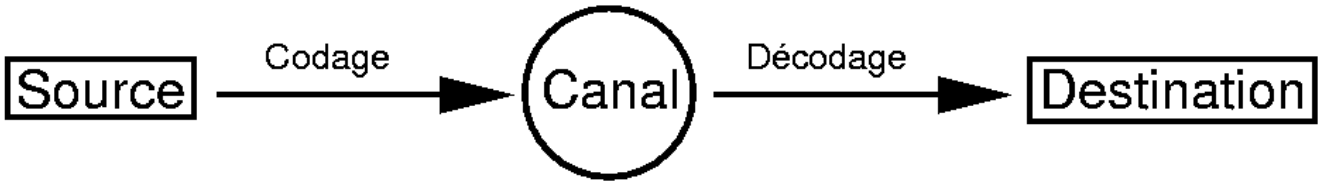
Taux d'erreur = Nombre de bits erronés sur le total des bits transférés

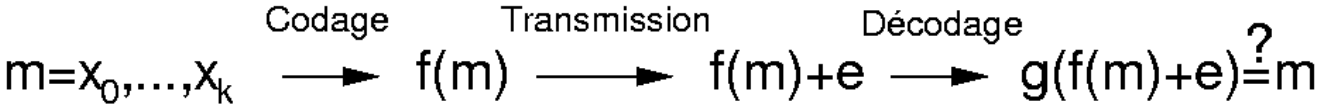
- Disquette magnétique : 1 bit erroné tous les milliards de bits transférés
  - Un million de bits/s (125 Ko/s) : 1 bit erroné toutes les 16.6 minutes
  - Lecteurs actuels (5 Mo/s) : 1 bit erroné toutes les 25 secondes
- CD-ROM optique : 1 bit erroné tous les 100 000 bits (12.5 Ko) transférés
  ⇒ 6300 erreurs dans un disque
- Audio DAT : 10^{-5} bits faux (à 48kHz) ⇒ 2 erreurs chaque seconde
- Ligne téléphonique : 10^{-4} à 10^{-6} bits erronés
- Communicateurs par fibres optiques : 10^{-9} bits erronés
- Mémoires à semi-conducteurs : $< 10^{-9}$

Plan du cours

- I. Introduction : Notion de code
  - Concepts de base de la correction: longueur et distance.
  - Exemples introductifs
    - Commande automatique d'un bras de robot
    - Contrôle de parité
    - Parité longitudinale et transversale
- Codes détecteurs d'erreur
- Généralisation : Code linéaire
  - Définition. Exemple : parité, codes de Hamming
  - Codes cycliques et Reed-Solomon
- Autres codes et applications

I. Notion de Code





- Le code doit répondre à différents critères :
  - Sécurité de l'information : cryptage + authentification
  - Rentabilité : compression des données
  - Tolérance aux fautes : correction/détection d'erreurs

Concepts de base de la correction

- Un groupe de bits dans un ordinateur est un « mot ».
- Chaque bit est considéré comme étant une « lettre ».

- La langue française nous permet une analogie :
  - Toutes les combinaisons possibles de l'alphabet ne sont pas des mots de la langue. Les seuls mots autorisés sont ceux énumérés dans un dictionnaire.
  - Des erreurs qui se produisent en transmettant ou en stockant des mots français peuvent être détectées en déterminant si le mot reçu est dans le dictionnaire.
  - S'il ne l'est pas, des erreurs peuvent être corrigées en déterminant quel mot français existant est le plus proche du mot reçu.

⇒ Idée pour la correction d'erreurs :
  - Ajouter des lettres supplémentaires (redondantes).
  - Ces lettres supplémentaires donnent une structure à chaque mot.
  - Si cette structure est changée par des erreurs, les changements peuvent être détectés et corrigés.

Commande automatique d'un bras de robot

Code	Haut	Bas	Droite	Gauche
C_2	00	10	01	11

- C_2 : économique
- ☹ Impossible de détecter une erreur :
  - Si 00 est envoyé et 01 reçu, « droite » est interprété au lieu de « haut »

Commande automatique d'un bras de robot

Code	Haut	Bas	Droite	Gauche
C_2	00	10	01	11
C_3	000	101	011	110

- C_3 : détecte si 1 seul bit est faux car 2 mots distincts diffèrent d'au moins 2 bits (distance de Hamming)
  - ☺ Si 000 est envoyé et 001 est reçu : erreur
  - ☹ Pas de correction : si 001 est reçu, avec une seule erreur il peut tout aussi bien provenir de 000 que 011 ou encore 101 !!!

Commande automatique d'un bras de robot

Code	Haut	Bas	Droite	Gauche
C_2	00	10	01	11
C_3	000	101	011	110
C_6	000000	111000	001110	110011

- C_6 : distance minimale entre deux mots : 3

  ⇒Détecte 2 erreurs

  ☺Corrige 1 erreur :
  - Avec au plus un bit erroné, on choisit le mot de code (du dictionnaire) le plus proche
  - Ex: 000001 est reçu, alors 000000 est le mot admissible le plus proche

Définition - Notation

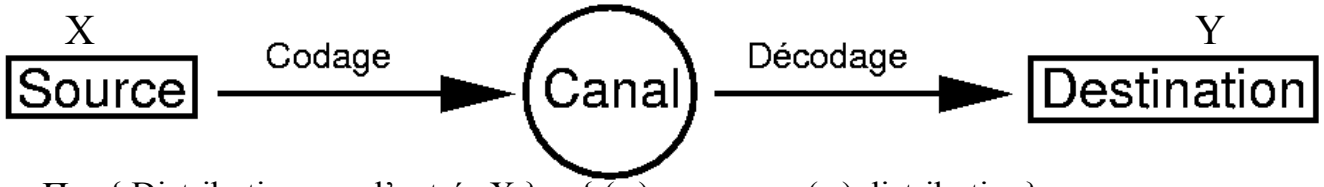
- **V = alphabet** = ensemble fini de symboles.
  Ex1: V= {0,1} Ex2: V={octets}

- Code de **longueur *n*** sur V = sous ensemble de V^n.
  - Les éléments du code sont appelés mots de code.

- **Codage** par blocs de source de taille ***k*** $(k < n)$
  - **Φ** : V^k --> V^n : fonction de codage, injective
  - $\Phi(x_1, \ldots, x_k) = y_1, \ldots, y_k, \ldots, y_n$
  - **r** = n – k = nombre de symboles de redondance
  - **Rendement**: **R = k/n** $(0< R \leq 1)$

- **Code(n, k) sur V** = sous-ensemble de V^n de cardinal $|V|^k$.

Lien avec entropie: capacité de canal



- $\underline{\Pi}$ = { Distributions sur l'entrée X } = { $(p_i)_{i=1..|V|}$ avec (p_i) distribution}
- $p_{i|k}$ = Prob ($Y = s_i \mid X=s_k$) : caractérise les probabilités d'erreurs lors de la transmission sur le canal sans mémoire.
  - Canal sans erreur ssi ($p_{i,i}=1$ et $p_{i,k\neq i}=0$)
- $P(Y=s_i) = \sum_{k=1..|V|} p_k.p_{i|k}$
- **Déf**: ***Capacité de canal*** : $C = Max_{p \in \underline{\Pi}} H(X) - H(X \mid Y)$
  - i.e. ce qu'il reste à découvrir de l'entrée X du canal lorsqu'on connait la sortie Y.
- On a aussi: $C = Max\ H(Y) - H(Y \mid X)$. Cas extrêmes:
  - $H(Y)=H(Y|X)$: sortie indépendante de l'entrée.
  - $H(Y \mid X) = 0$: canal sans erreur.

Deuxième théorème de Shannon

- **Théorème**: Soit un canal de capacité C. Alors pour tout $\varepsilon>0$:
  $\exists$ un code(n, k) de probabilité d'erreur $<\varepsilon$ ssi $0 \leq k/n < C$.

  i.e. la capacité de canal C est une limite supérieure au rendement..

- **Exemple : canal binaire symétrique (BSC).**
  - $C_{BSC} = 1 + p.\log_2 p + (1-p).\log_2(1-p) = 1 - H(p)$
  - Si $p = 0.5$ => $C_{BSC} = 0$: pas de code correcteur possible.
  - Si $p \neq 0.5$ => il existe un code permettant de communiquer sans erreur
    - Mais son rendement est borné par C.
    - Exemples: p= 0,8 => R < 27% ; p=0,9=> R < 53% ; p=0,99 => R < 92%
- **Problème fondamental du codage:**
  - Construire des codes de rendement maximal pour une longueur n fixée.

PREMIERS EXEMPLES
Codes de Parité

- Code ASCII
- Codes de Parité longitudinale et transversale
- Exemples de codes de parité usuels
- Codes de Hamming

Contrôle de parité

- Une technique de base pour construire un code détecteur
  1. Découper le message en mots de 7 bits $m=[x_0,\ldots, x_6]$
  2. Ajouter aux mots leur parité : $f(m)=[x_0,\ldots, x_6, p]$
     - Le nombre de 1 dans le mot est soit pair (p = 0) soit impair (p = 1)
     - Calculée par : $\mathbf{x_7 = p = \sum_{i=0..6} x_i \bmod 2}$

- Standard n°5 du Comité Consultatif International Télégraphique et Téléphonique (CCITT 5) : le plus populaire, utilisé par exemple aux USA.

Lettre	Codage de base sur 7 bits	Mot de code avec bit de parité
a	1000 001	1000 001**0**
e	1010 001	1010 001**1**
u	0110 101	0110 101**0**

☺ Permet de détecter tout nombre impair d' erreurs

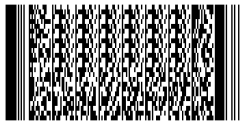
« Parités » usuelles pour la simple détection d'erreur

- LUHN10 pour les cartes bleues : dernier chiffre = chiffre de parité
  - Doubler modulo 9 un chiffre sur deux du n°
    - Exemple : 4561 0032 4001 236**c**
    - (4*2%9)+5+(6*2%9)+1+(0*2%9)+0+(3*2%9)+2+ (4*2%9)+0+(0*2%9)+1+ (2*2%9)+3+(6*2%9)
      = 8 +5+ 3 +1+ 0 +0+ 6 +2+ 8 +0+ 0 +1+ 4 +3+ 3 =44
  - Le résultat doit être 0 modulo 10 pour une carte valide
    - Donc c = 10 – (44 % 10) = 6 => n° valide = 4561 0032 4001 2366
- Clefs (sécurité sociale, RIB, etc.)
    - Sécu : clef calculée pour le numéro + la clef soit nul modulo 97
    - RIB : clef calculée pour que (numéro||clef)$_{5+5+11+2 \text{ chiffres}}$ soit nul modulo 97
  - IBAN : lettres + 9 et la somme doit faire 1 modulo 99
- De 1972 à 2077: Code ISBN sur les livres sur 10 chiffres :
  - $\sum_{i=1..10}$ i £ a_i ´ 0 modulo 11

Code barre EAN-13

- EAN-13 (European Article Numbering)
- Numéro sur 13 chiffres + motif graphique barres noires/blanches

  c_{12} - c_{11} .. c_6 - c_5 .. c_0
- c_0 chiffre de parité calculé comme suit:
  - Soient a = mod 10
    etb = $c_{11} + c_9 + c_7 + c_5 + c_3 + c_1$ mod 10
  - Alors c_0 = 10 - (a+3b mod 10)
  - Exemple: a= 3+9+1+3+5+7 mod 10 = 8 ; b= 2+9+2+4+6+8 mod 10 = 2; c_0 = 10 - (a+3b mod 10) = 10 - 4 = 6
- Le code barre graphique code le même numéro:
  - chaque colonne de 2,31mm code un seul chiffre, par 4 barres de largeur différentes chaque colonne est divisée en 7 barres N/B de largeur élémentaire 0,33 mm
- EAN13 permet de détecter des erreurs mais pas de corriger.
- Depuis 2007: Code ISBN sur les livres=EAN-13: $c_{12}c_{11}c_{10}c_9c_8c_7c_6c_5c_4c_3c_2c_1c_0$
  - Avec pour les livres: $c_{12}c_{11}c_{10}$=978
    - Ex: **978-2-10-050692-7**
      c_0=10-[9+8+1+0+0+9+3x(7+2+0+5+6+2) mod 10]=10- 3 = 7
- Extensions: code barre bidimensionnel PDF417: permet de coder jusqu'à 2725 caractères, grâce à un code correcteur de Reed-Solomon



Parité longitudinale et transversale

a_{00}	a_{01}	a_{02}	a_{03}	a_{04}	a_{05}	a_{06}	P_0
a_{10}	a_{11}	a_{12}	a_{13}	a_{14}	a_{15}	a_{16}	P_1
a_{20}	a_{21}	a_{22}	a_{23}	a_{24}	a_{25}	a_{26}	P_3
C_0	C_1	C_2	C_3	C_4	C_5	C_6	N

1. Mots sur 3*7 = 21 bits
2. Parité par ligne : P_i
3. Parité par colonne : C_j
4. Parité globale : N

- Détecte 2 ou tout nombre impair d'erreurs
- Corrige 1 erreur
  - Un des a_{ij} est faux : le recalcul de P_i et C_j donne l'emplacement ij
  - P_i, C_j et N sont recalculés
- Détecte si il y a 2 erreurs, mais ne permet pas de corriger

Rendement

- Nombre de bits de message / Nombre de bits transmis
  - Parité : Rendement = 7/8 = 87.5 %
  - Parité long. & transv. : Rendement = 21/32 ≈ 65 %

Distance de Hamming, Taux de correction, Bornes supérieures

Distance d'un code

- Codage: bijection $\Phi : V^k \dashrightarrow C$ avec C inclus dans V^n
  - $\Phi(x_1, \ldots, x_k) = y_1, \ldots, y_k, \ldots, y_n$
- Définitions:
  - **Distance de Hamming** dans V^n: $d_H(x,y) = \text{Card}\ \{ i / x_i \neq y_i \}$
  - **Distance du code C** : $d = \text{Min}\ \{ d_H(x, y) ; \forall x,y \in C \}$
- A la sortie du canal, on reçoit $z \in V^n$:
  - Si $\mathbf{z \in C}$: pas d'erreur détectée; décodage en calculant $\Phi^{-1}(z_1, \ldots, z_n)$.
  - Sinon, $\mathbf{z \notin C}$: on **détecte qu'il y a erreur(s)** dans les symboles reçus !
    On corrige z en y avec y=mot de C le plus proche de z: $d_H(y,z)=\min_{c\in C} d_H(c, z)$.
- **Théorème** : si C est un code de distance d, alors:
  - on **détecte** jusqu'à **(d-1)** erreurs de symboles par mot de code;
  - on corrige jusqu'à $\lfloor$ **(d-1)/2** $\rfloor$ erreurs de symboles par mot de code.

Codes équivalents, étendus raccourcis

- C : code (n,k,d) sur V
- Def 1: code **équivalent** : C' obtenu par
  - Permutation de positions dans tous les mots de C
  - Permutation de symboles de V dans tous les mots de C

  C' a même distance et rendement que C
- Def 2: code **étendu** : C' obtenu par ajout d'un chiffre de parité:

  $C' = \{ c_1 \ldots c_n\ c_{n+1}\ \text{tq}\ c_1 \ldots c_n \in C\ \text{et}\ c_1 + \ldots + c_{n+1} = 0 \}$
- Def 3: code **poinçonné** : C' obtenu en supprimant une position:

  $C' = \text{poinçonné}(C,i) = \{ c_1 \ldots c_{i-1}\ c_{i+1} \ldots c_n\ \text{tq}\ c_1 \ldots c_n \in C \}$

  si on poinçonne m positions: $d' \geq d-m$
- Def 4 : code **raccourci** : soit $s \in V$ et $1 \leq i \leq n$ fixés:

  $C' = \text{raccourci}(C,i,s) = \{ c_1 \ldots c_{i-1}\ c_{i+1} \ldots c_n\ \text{tq}\ c_1 \ldots c_{i-1}\ s\ c_{i+1} \ldots c_n \in C \}$

  on a $d' \geq d$

Exemple: Code de Hamming

Code binaire 1-correcteur

- Code binaire de longueur n:
  - Si il y a 0 ou 1 erreur de bits => n+1 possibilités
- Au moins r = $\log_2$ (n+1) bits pour coder une erreur possible
- **Code de Hamming**: code (2^r-1, k=2^r-r-1, d=3)
  $c_1 \ldots c_n$ défini par :
  - Si i ≠ 2^j alors c_i est un bit de source
  - Si i = 2^j alors c_i est un bit de contrôle de parité = somme des c_k tel que k écrit en binaire a un 1 en position j
  - Exemple code de Hamming (7,4)
- Détection: si au moins un des bits de parité est erroné
- Correction: on change le bit d'indice la somme des indices des bits de parité erronés

Code de Hamming

- Code 1-correcteur à nombre de bits ajoutés minimal, $\delta = 3$

🔔 Idée : ajouter un contrôle de parité pour chaque puissance de 2
➔ b1, b2, b4, b8, b16, etc.

⇒ Cela suffit pour localiser l'erreur !

⇒ $C(n, n-\lfloor \log_2(n) \rfloor -1)$ → rendement $\approx 1-\log_2(n)/n$

$$[x_0 \quad x_1 \quad x_2 \quad x_3] \begin{matrix} b_1 & b_2 & & b_4 & & & \\ \end{matrix} \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

⇒ rendements :
- 4/7 ≈ 57 %
- 11/15 ≈ 73 %
- 26/31 ≈ 84 %

…

Code de Hamming

- Théorème:
  - Les codes de Hamming $C(n, n-\lfloor \log_2(n) \rfloor -1)$ sont de distance 3 :
    - 1-correcteurs
    - Tout mot reçu est corrigé avec 0 ou 1 erreur (code 1-parfait)
  - Tout code binaire linéaire 1-correcteur parfait est un code de Hamming.

- Code de Hamming étendu C' $(n+1, n-\lfloor \log_2(n) \rfloor -1)$
  - Ajout d'un bit de parité = xor des n bits
  - Distance 4 (exercice) donc corrige 1 erreur et 1 effacement.

- Rem.: code binaire **étendu** : obtenu par ajout d'un bit de parité:

$$C' = \{ c_1 \dots c_n \; c_{n+1} \text{ tq } c_1 \dots c_n \in C \text{ et } c_1 + \dots + c_{n+1} = 0 \}$$

Propriété: Si d est impaire, alors C' est de distance d'=d+1.

Application: Code du minitel

- Faible taux d'erreur, mais paquets longs :
  - Code de Hamming + detection paquets
- Source = suite de paquets de 15 octets = 120 bits
- Correction d'1 erreur : Code Hamming(127,120,3) :
  - 7 bits + 1 bit parité pour les 7 bits contrôle = 1 octet
- Détection de paquets: 1octet avec que des 0
  - Si erreur détectée : ARQ
- Total : 17 octets, rendement = 15/17 = 88%

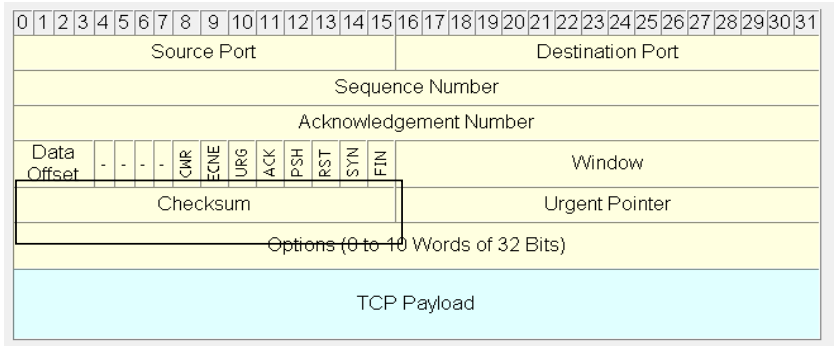
Plan du cours

- Introduction : Notion de code
- Définition, distance, effacements et erreurs
- Code de Hamming
- **Codes détecteurs d'erreur**
  - LFSR et polynômes.
    Corps de Galois.
  - Codes CRC
  - Propriétés. Applications
- Codes correcteurs : Code linéaire, Reed Solomon
  - Codes cycliques et Reed-Solomon
- Autres codes et applications
  - Rafales d'erreurs. Code CIRC.

Exemple: contrôle de parité dans TCP

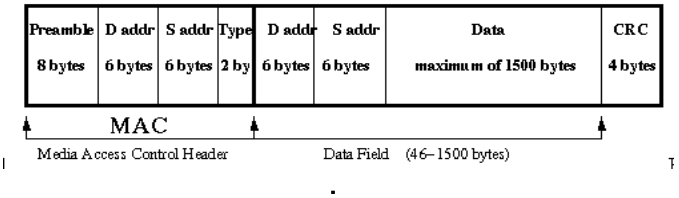
7	Application (HTTP,FTP, DNS)
4	Transport (TCP, UDP)
3	Network (IP)
2	Data Link (Ethernet, 802.11b)
1	Physical

TCP Packet Format



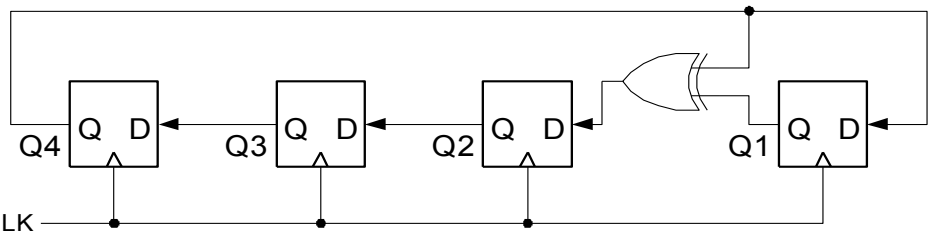
• TCP Checksum :bits de contrôle, calculés par l'émetteur à partir de l'en-tête et des données, et vérifiés lors de la réception.

Exemple: Ethernet CRC-32



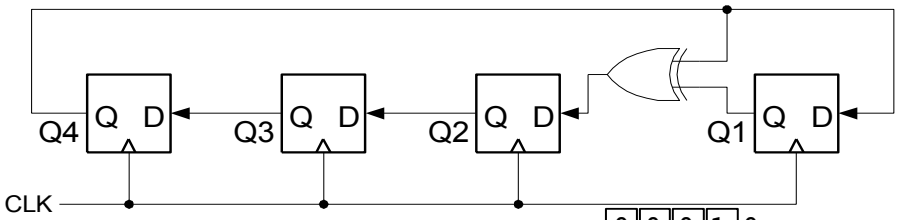
Linear Feedback Shift Registers (LFSRs)

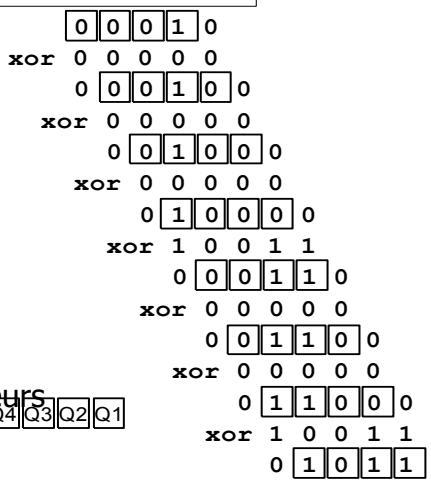
- LFSR : circuit élémentaire à base de registres à décalage et quelques xor.
- Exemple 4-bit LFSR :



- Avantages:
  - Hardware très simple => opération rapide
  - Rapide (débit élevé) : en général, nombre faible de xors (2 !)
- Applications:
  - générateurs pseudo-aléatoires (non cryptographique)
  - compteurs
  - chiffrement (pas seuls ! avec des NLFSRs, des Sbox etc)
  - **détection et correction d'erreurs**

4-bit LFSR

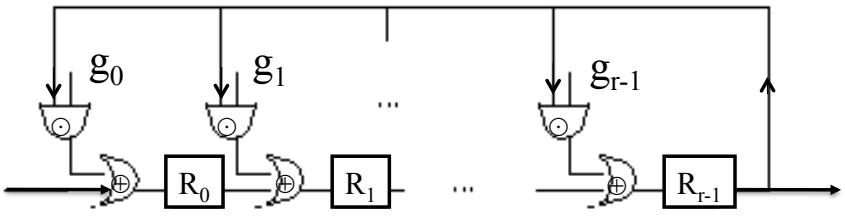




0001
0010
0100
1000
0011
0110
1100
1011
0101
1010
0111
1110
1111
1101
1001
0001

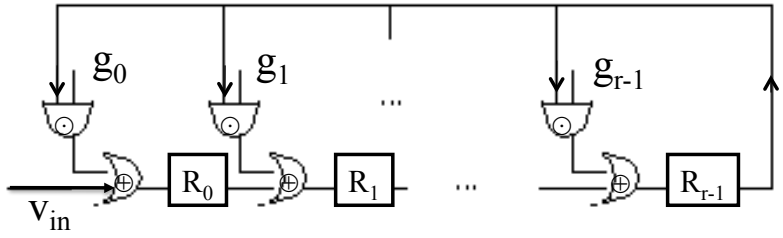
- Etat = valeur des registres [Q4,Q3,Q2,Q1]
- Etat à t+1 = "Left-shift circulaire" état à t
  suivi de xor avec bit le plus à gauche
- Exemple:
  - Q4(t+1) = Q3(t)
    Q3(t+1) = Q2(t)
    Q2(t+1) = Q1(t) + Q4(t)
    Q1(t+1) = Q4(t)
- En général, avec n bascules, cycle sur 2^n-1 valeurs différentes non nulles :
  - Exemple: 15 valeurs (NB 0000 est absorbant)
  - générateur pseudo-aléatoire avec une longue période.

Généralisation: LFSR (dans un corps)



- Valeurs des registres du LFSR :
  - à t : $[R_{r-1}, R_{r-2}, \ldots R_0]$
  - à t+1 : $[R'_{r-1}, R'_{r-2}, \ldots R'_0]$ avec $R'_i = R_{i-1} + g_i.R_{r-1}$
- Polynôme caractéristique: $g(X) = X^r - g_{r-1}X^{r-1} - \ldots - g_0X^0$.
- Notation: K[X] = ensemble des polynômes à coefficients dans K

Généralisation: LFSR avec opérations dans K. Lien avec polynôme dans K[X]



- Valeurs des registres du LFSR :
  - à t : $[R_0, R_1, \ldots R_{r-1}]$
  - à t+1 : $[R'_0, R'_1, \ldots R'_{r-1}]$
    avec $R'_i = R_{i-1} + g_i.R_{r-1}$
- Polynôme caractéristique :
  - $g(X) = X^r - g_{r-1}X^{r-1} - \ldots - g_0X^0$.

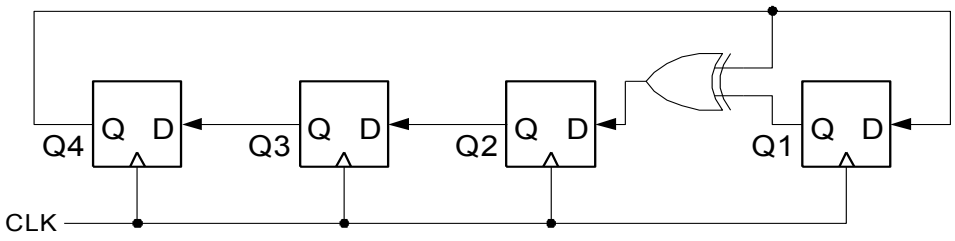
- Valeurs des registres du LFSR, vues comme un polynôme :
  - à t : $[R_0, R_1, \ldots R_{r-1}]$: $P_t(X) = R_{r-1}.X^{r-1} + R_{r-2}.X^{r-2} + \ldots + R_0.X^0$
  - à t+1: $[R'_0, R'_1, \ldots R'_{r-1}]$:

$$\begin{aligned} P_{t+1}(X) &= R'_{r-1}.X^{r-1} + R'_{r-2}.X^{r-2} + \ldots + R'_0.X^0 \\ &= (R_{r-2} + g_{r-1}.R_{n-1})X^{r-1} + \ldots + (R_{i-1} + g_i.R_{n-1})X^i + \ldots + (v_{in} + g_0.R_{n-1})X^0 \\ &= X.P_t(X) - R_{n-1}.g(X) + v_{in}.X^0 \\ &= [\, X.P_t(X) \ \textit{mod}\ g(X) \,] + v_{in}.X^0 \end{aligned}$$

- Si K=corps : Reste *division euclidienne polynômiale* par g(X) dans K[X]

Cas binaire

- $G(X) = X^4 + X + 1$



Plan du cours

- Introduction : Notion de code
- Définition, distance, effacements et erreurs
- Code de Hamming
- **Codes détecteurs d'erreur**
  - LFSR et polynômes.
  - **Corps de Galois.**
  - Codes CRC
  - Propriétés. Applications
- Codes correcteurs : Code linéaire, Reed Solomon
  - Codes cycliques et Reed-Solomon
- Autres codes et applications
  - Rafales d'erreurs. Code CIRC.

Corps de Galois et Opérations + et . d'un LFSR

- Un **corps** (K, +, x, "0", "1") est défini par:
  - deux opérations internes :
    - + "addition" et x "multiplication"
  - associatives et distributivité
  - Éléments neutres: "0" pour +, "1" pour x.
  - Chaque élément u a un opposé pour + : $-u$
    $u-u=$"0"
  - Chaque élément u non nul a un inverse pour x : u^{-1}
    $u \times u^{-1}=$"1"
- Les corps finis sont appelés "corps de Galois".
- Il existe un unique corps fini avec q éléments ssi $q = p^m$ avec p premier et $m \geq 1$, noté GF(q) ou F_q.

- Exemple de corps:
  - Q = ensemble des rationnels
  - R = ensemble des réels
  - Z/pZ avec p premier
  - C = R[i]/i^2+1
- Exemples de corps finis:
  - GF(p) = Z/pZ (p premier)
  - GF(2) = ({F,T}, xor, and, F, T)
  - GF(256) = corps des "octets"

- GF(p^m) est isomorphe à GF(p)/g(X) avec g(x) *polynôme primitif* de GF(p)[X]
  - Addition = vectorielle coef à coef
  - Multiplication = LFSR(g)
  - Exemple GF(256) : + = xor bit à bit ; x = LFSR($X^8 + X^4 + X^3 + X^1 + X^0$)

Corps finis

- Propriétés élémentaires

Corps finis – Propriétés élémentaires (1/3)

- **Notation**: **F[X]** = anneau des polynômes à 1 variable et à coefficients dans le corps F.
  Un polynome est dit **unitaire** ssi son coefficient de tête est 1.

- **Définition**: Soit $Q(X) \in F[X]$;
  Q est **irréductible** ssi $Q(X)=A(X).B(X)$ avec A(X) et B(X) dans F[X] implique degré(A)=0 ou degré(B)=0.

- **Théorème**: F[X] est **factoriel** (i.e. tout polynôme unitaire de F[X] s'écrit de manière unique comme produit de polynômes unitaires irréductibles)

- **Théorème** (fondamental de l'algèbre): soit $Q(X) \in F[X]$ de degré d;
  alors Q(X) admet au plus d racines dans F.

Corps finis – Propriétés élémentaires (2/3)

- Pour tout entier premier p, l'ensemble des entiers mod p est un corps noté $\mathbf{F}_p$: $\mathbf{F}_p = (\{0, 1, \ldots, p\text{-}1\}, +_{\text{mod } p}, *_{\text{mod } p}, 0, 1)$

- **Théorème:** Soit h(X) un polynôme irrédictible de degré m : l'anneau quotient F[X]/(h(X)) est un corps fini avec $|F|^p$ éléments.
  $F[X]/(h(X)) = (\{ \Sigma_{i=0}^{m-1} a_i X^i ; a_i \in F \}, +_{\text{mod } h(X)}, *_{\text{mod } h(X)}, 0.X^0, 1.X^0)$

  Remarque: similaire à $F_p = Z/(p)$

- Le corps F[X]/(h(X)) est une extension du corps F de degré m. C'est aussi un espace vectoriel de dimension m sur F.

Corps finis – Propriétés élémentaires (3/3)

- **Théorème:** Tout corps fini est de cardinal $q=p^m$ et est isomorphe à F_p si m=1; $F_p[X]/(h(X))$ sinon, avec $h \in F[X]$ irrédcutible de degré m.

  Réciproquement: Pour tout $q=p^m$, il existe (à un isomorphisme près) un unique corps de cardinal q, noté F_q.

- **Théorème:** $F_q^* = (\{x \in F_q; x \neq 0\}, *)$ est un groupe cyclique. Un générateur γ de F_q^* est appelé *élément primitif* :
  $F_q^* = \{ 1, \gamma, \gamma^2, \ldots, \gamma^{q-2} \}$.

- **Théorème:** Il y a $\varphi(q-1)$ éléments primitifs.
  - $\varphi(q-1) = \text{Card}\{ \text{i entier; } 1 \leq i < q-1 \text{ et i premier avec } q-1\}$ [indicatrice d'Euler].

Exemple : GF(16) avec LFSR (X4+X+1)

$\alpha^0 = 1$
$\alpha^1 = x$
$\alpha^2 = x^2$
$\alpha^3 = x^3$
$\alpha^4 = x + 1$
$\alpha^5 = x^2 + x$
$\alpha^6 = x^3 + x^2$
$\alpha^7 = x^3 + x + 1$
$\alpha^8 = x^2 + 1$
$\alpha^9 = x^3 + x$
$\alpha^{10} = x^2 + x + 1$
$\alpha^{11} = x^3 + x^2 + x$
$\alpha^{12} = x^3 + x^2 + x + 1$
$\alpha^{13} = x^3 + x^2 + 1$
$\alpha^{14} = x^3 + 1$
$\alpha^{15} = 1$

- X4 + X + 1 est un polynôme primitif.

.

$\alpha^4 = x^4 \bmod x^4 + x + 1$
$= x^4 \text{ xor } x^4 + x + 1$
$= x + 1$

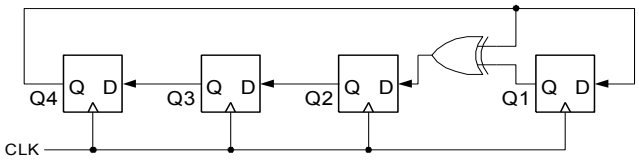


Table de polynômes primitifs

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

Galois Field		Hardware
Multiplication by x	$\Leftrightarrow$	shift left
Taking the result mod $p(x)$	$\Leftrightarrow$	XOR-ing with the coefficients of $p(x)$ when the most significant coefficient is 1.
Obtaining all 2^n-1 non-zero elements by evaluating x^k for $k = 1, \ldots, 2^n-1$	$\Leftrightarrow$	Shifting and XOR-ing 2^n-1 times.

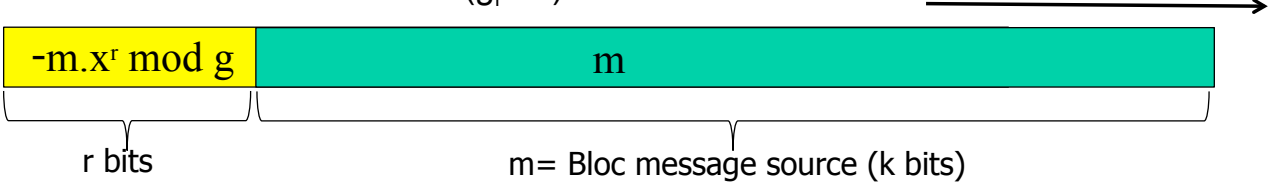
Plan du cours

- Introduction : Notion de code
- Définition, distance, effacements et erreurs
- Code de Hamming
- **Codes détecteurs d'erreur**
  - LFSR et polynômes.
  - Corps de Galois.
  - **Codes CRC**
  - Propriétés. Applications
- Codes correcteurs : Code linéaire, Reed Solomon
  - Codes cycliques et Reed-Solomon
- Autres codes et applications
  - Rafales d'erreurs. Code CIRC.

Codes réseaux informatiques : codes CRC (1)

- **CRC** : **cyclic redundancy check** avec r bits de redondance
  Polynome générateur $g(X) = x^r + \Sigma_{i=0..r-1}\ g_i.x^i$ de degré r sur F_2

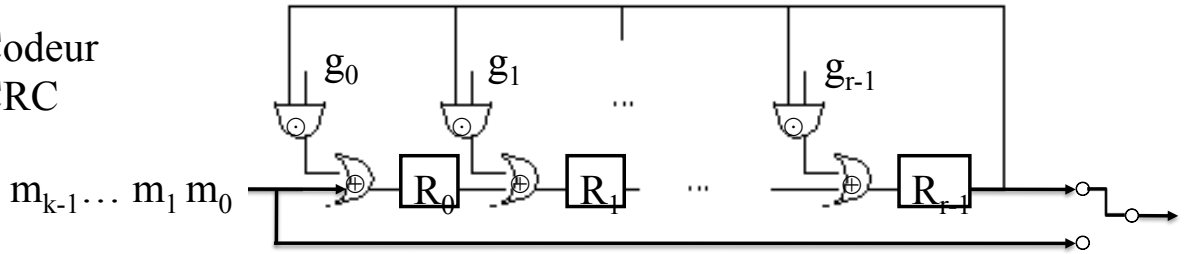
- **Mots de code** : les multiples de g(X)
  - Source: m=k bits ⇔ m(x) polynome de degré k-1
    - Exemple: m=01101 $m(x) = x+x^2+x^4$
  - Codage: $c = m.x^r - (m.x^r \bmod g) = m.x^r + (m.x^r \bmod g)$
    - On a c multiple de g
    - Redondance r bits ($g_r = 1$)



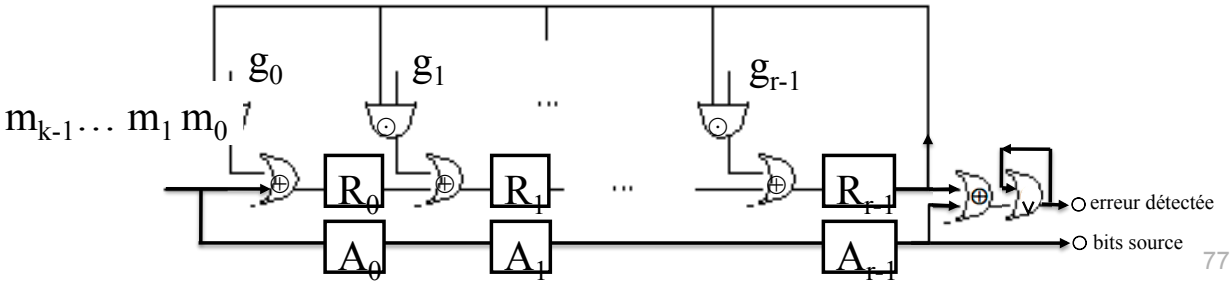
- Décodage : détection d'erreurs
  - Réception de y et vérification que y(x) mod g(x) = 0 !

Codeur et décodeur CRC par LFSR

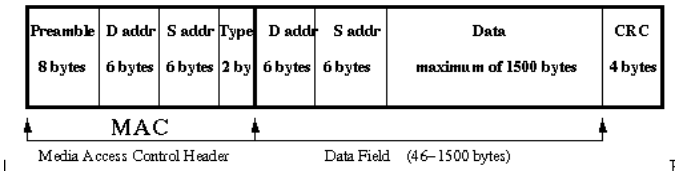
Codeur CRC



Décodeur CRC



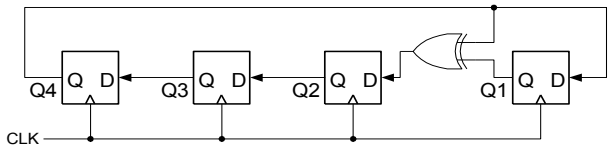
Example: Ethernet CRC-32



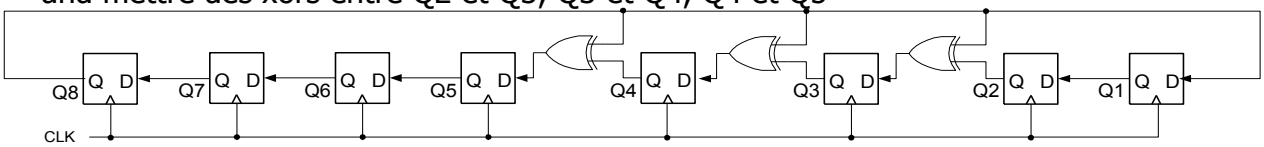
Construction LFSR pour polynôme primitif

- Pour un LFSR à k-bit, numéroter les registres-bascules avec Q1 à droite.
- Le feedback vient du registre le plus à gauche (Qk).
- Choisir un polynôme primitif de la forme $x^k + \ldots + 1$.
- Chaque monôme de la forme $1.x^i$ correspond à un xor entre Qi et Q$i+1$.

- Exemple 4-bit: utiliser $x^4 + x + 1$
  - x^4 ⇔ sortie de Q4
  - x ⇔ xor entre Q1 et Q2
  - 1 ⇔ entrée de Q1



- Pour un LFSR 8-bit, utiliser le polynôme primitif $x^8 + x^4 + x^3 + x^2 + 1$ and mettre des xors entre Q2 et Q3, Q3 et Q4, Q4 et Q5



Construction de codes CRC : propriétés pour le choix du polynôme.

- **Propriétés**: un code CRC binaire détecte :
  - si $g_0 = 1$, détecte les erreurs de poids 1
  - si g(x) a un facteur avec au moins 3 termes: détecte les erreurs de poids 2.
  - si g a (x+1) comme facteur: détecte un nombre impair d'erreurs
  - Si g(X)=p(x).q(x) avec p polynome **primitif** de degré d
    - Détecte toute erreur sur 2 bits distants d'au plus 2^d-1 bits consécutifs
    - Souvent, on choisit g(x)=(X+1).p(X) avec p primitif : détecte
      - tout paquet de taille inférieure à deg(g)
      - 1, 2 ou 3 erreurs isolées (si n < 2^r pour détecter 2 erreurs)
- Question: quelles propriétés sont vraies si V = GF(q) ?

Exemples de CRC standards

Nom	Générateur	Factorisation	Exemples d'utilisation
TCH/FS -HS-EFS	$X^3 + X + 1$	irréductible	GSM transmission de voix
GSM TCH/EFS	$X^8 + X^4 + X^3 + X^2 + 1$	irréductible	GSM pré-codage canal à plein taux
CRC-8	$X^8 + X^7 + X^4 + X^3 + X + 1$	$(X + 1)(X^7 + X^3 + 1)$	GSM 3ème génération
CRC-16 X25-CCITT	$X^{16} + X^{12} + X^5 + 1$	$(X + 1)(X^{15} + X^{14} + X^{13} + X^{12} + X^4 + X^3 + X^2 + X + 1)$	Protocole X25-CCITT ; contrôle trames PPP FCS-16 (RFC-1662)
CRC-24	$X^{24} + X^{23} + X^{18} + X^{17} + X^{14} + X^{11} + X^{10} + X^7 + X^6 + X^5 + X^4 + X^3 + X + 1$	$(X + 1)(X^{23} + X^{17} + X^{13} + X^{12} + X^{11} + X^9 + X^8 + X^7 + X^5 + X^3 + 1)$	communications UHF et satellites (SATCOM) ; messages OpenPGP (RFC-2440)
CRC-24 (3GPP)	$X^{24} + X^{23} + X^6 + X^5 + X + 1$	$(X + 1)(X^{23} + X^5 + 1)$	GSM 3ème génération
CRC-32 AUTODIN-II	$X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$	irréductible	IEEE-802.3, ATM AAL5, trames PPP FCS-32 (RFC-1662) ; contrôle d'intégrité des fichiers ZIP et RAR ;

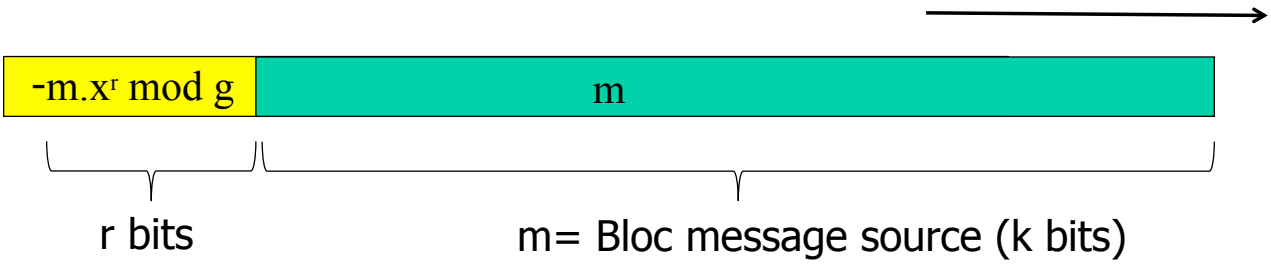
- CRC-16: $G(x) = x^{16} + x^{15} + x^2 + 1 = (X+1)(X^{15} + \ldots)$
  - Détecte 1, 2 erreurs
  - Toutes les erreurs sur un nombre impair de bits
  - Paquets d'erreurs de longueur ≤ 16
  - La plupart des paquets plus longs
- CRC-32: $G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$
  - Utilisé dans ethernet
  - Il y a 32 bits àf 1 ajouté en tête de message de message
    - Initialisation des registres du LFSR à 1.

Codes réseaux informatiques : codes CRC (1)

- **CRC** : **cyclic redundancy check** avec r bits de redondance
  Polynome générateur $g(X) = x^r + \Sigma_{i=0..r-1}\ g_i.x^i$ de degré r sur F_2

- Source: m=k bits ⇔ m(x) polynome de degré k-1
  - Exemple: m=01101 $m(x) = x+x^2+x^4$

- Codage: $c = m.x^r - (m.x^r \bmod g) = m.x^r + (m.x^r \bmod g)$
  - On a c multiple de g
  - Redondance r bits ($g_r = 1$)
  - Intérêt: détecte tout paquet d'erreurs portant sur au plus r bits !



Codes réseaux informatiques : codes CRC (2)

- Propriétés: un code CRC binaire détecte :
  - erreurs de poids 1 si : $g_0 = 1$
  - erreurs de poids 2 si g(x) a un facteur avec au moins 3 termes
  - nombre impair d'erreurs si g a (x+1) comme facteur
  - Si g(X)=p(x).q(x) avec p polynôme **primitif** de degré d
    - Détecte toute erreur sur 2 bits distants d'au plus 2^d-1 bits consécutifs
    - Souvent, on choisit g(x)=(X+1).p(X) avec p primitif : détecte
      - tout paquet de taille inférieure à degré(g)
      - 1, 2 ou 3 erreurs isolées (si $n < 2^r$ pour détecter 2 erreurs)

- Question: quelles propriétés sont vraies si V = GF(q) ?

Exemples de CRC standards

Nom	Générateur	Factorisation	Exemples d'utilisation
TCH/FS -HS-EFS	X^3+X+1	irréductible	GSM transmission de voix
GSM TCH/EFS	$X^8+X^4+X^3+X^2+1$	irréductible	GSM pré-codage canal à plein taux
CRC-8	$X^8+X^7+X^4+X^3+X+1$	$(X+1)(X^7+X^3+1)$	GSM 3ème génération
CRC-16 X25-CCITT	$X^{16}+X^{12}+X^5+1$	$(X+1)(X^{15}+X^{14}+X^{13}+X^{12}+X^4+X^3+X^2+X+1)$	Protocole X25-CCITT ; contrôle trames PPP FCS-16 (RFC-1662)
CRC-24	$X^{24}+X^{23}+X^{18}+X^{17}+X^{14}+X^{11}+X^{10}+X^7+X^6+X^5+X^4+X^3+X+1$	$(X+1)(X^{23}+X^{17}+X^{13}+X^{12}+X^{11}+X^9+X^8+X^7+X^5+X^3+1)$	communications UHF et satellites (SATCOM) ; messages OpenPGP (RFC-2440)
CRC-24 (3GPP)	$X^{24}+X^{23}+X^6+X^5+X+1$	$(X+1)(X^{23}+X^5+1)$	GSM 3ème génération
CRC-32 AUTODIN-II	$X^{32}+X^{26}+X^{23}+X^{22}+X^{16}+X^{12}+X^{11}+X^{10}+X^8+X^7+X^5+X^4+X^2+X+1$	irréductible	IEEE-802.3, ATM AAL5, trames PPP FCS-32 (RFC-1662) ; contrôle d'intégrité des fichiers ZIP et RAR ;

III. Codes linéaires

- Hypothèse sur V = vocabulaire source : *corps*
- Code linéaire - caractérisation
- Codage et décodage d'un code linéaire

Généralisation : Code linéaire

- Code correcteur (n,k): $[x_0,\ldots,x_{k-1}] \rightarrow f([x_0,\ldots,x_{k-1}])=[y_0,\ldots,y_{n-1}]$
- f : fonction d'un ensemble V^k vers un ensemble V^n

🔔 Idée : si f est linéaire, alors les opérations de codage/décodage se font en temps linéaire/taille de message

⇒ Rapide (proportionnel à la taille du message)

⇒ Il faut : V^k, V^n espaces vectoriels donc V un corps

☺ Opérations modulo 2 (ex: parité) : $V = Z/_{2Z}$ est un corps !

☺ On travaille en général avec V à 2, 2^8 ou 2^{256} éléments

⇒ Alors f linéaire correspond à une matrice G (génératrice) et f(m)=m G :

$$[y_0 \quad y_1 \quad \cdots \quad y_{n-1}] = [x_0 \quad \cdots \quad x_{k-1}] \begin{bmatrix} g_{0;0} & g_{0;1} & \cdots & g_{0;n-1} \\ \vdots & \vdots & \vdots & \vdots \\ g_{k-1;0} & g_{k-1;1} & \cdots & g_{k-1;n-1} \end{bmatrix}$$

Exemple : la parité

$$[x_0 \quad x_1 \quad x_2 \quad (x_0 + x_1 + x_2 \bmod 2)] = [x_0 \quad x_1 \quad x_2] \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

Temps de calcul

☺ Pour tout code linéaire C(n,k), il existe une matrice normalisée G' = [I_k | T] qui engendre le même code

⇒ $[y_0,y_1,\ldots,y_{n-1}] = [x_0,x_1,\ldots,x_{k-1},b_k,\ldots,b_{n-1}]$

1. Codage : y = x G' (temps quadratique)
2. Décodage : x = premiers bits de y (immédiat)
3. Détection : Si $H=[T^t|-I_{n-k}]$, alors z erroné ssi $Hz \neq 0$!!! (quadratique)
4. Correction : table précalculée des *e* de poids min. tels que $He=Hz\neq0$ ➔ y=z-e est le mot correct le plus proche de z (temps constant)

Hypothèse sur le vocabulaire V du canal

- V est muni d'une structure de **corps fini** :
  soient a,b ∈ V : e_0, e_1, a+b, -a, a*b, a/b ∈ V

- Possible ssi **|V| = p^m** avec **p** premier
  Soit q = p^m : V est alors isomorphe à GF_q
  - Exemple: V= {0,1} ; V={mots de 32 bits}

- Implantation
  - GF_q isomorphe à (Z/pZ[x])/Q(x) avec Q polynôme irréductible de degré m à coefficients dans Z/pZ
  - NB avec p= 2: facile à implanter avec des registres à décalage !

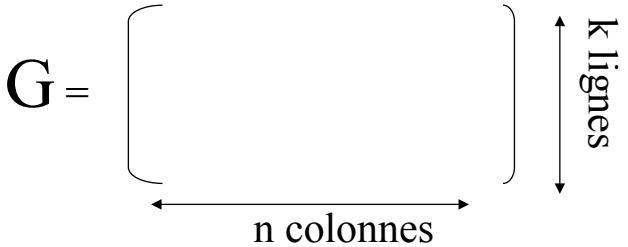
Code linéaire

- V corps => V^n est un espace vectoriel

- **Définition** : code linéaire ⇔ sev de V^n de dim k
  - Si $x=[x_0, \ldots, x_{n-1}] \in C$, $y=[y_0, \ldots, y_{n-1}] \in C$
    => $x+y=[x_0+y_0, \ldots, x_{n-1}+y_{n-1}] \in C$

- C est engendré par une **matrice génératrice** G

- C=Im(G)



Exemple 1

- V = {a=00, b=01, c=10, d=11}
- Code (6,4) engendré par G

G =

01	10	11	10	11	00
10	11	10	00	01	10
10	01	11	10	11	10
01	10	11	11	10	01

Exemple 2: code de Hamming (7,4)

- C' est aussi un code linéaire sur V = {0,1}

engendré par G =

1	1	1	0	0	0	0
1	0	0	1	1	0	0
0	1	0	1	0	1	0
1	1	0	1	0	0	1

- Remarque: il est équivalent aux codes de matrices génératrices G' =L.G.P avec L inversible et P permutation. Par exemple:

G'=

1	0	0	0	1	1	0
0	1	0	0	1	0	1
0	0	1	0	0	1	1
0	0	0	1	1	1	1

G''=

1	0	1	0	1	0	0
0	1	0	0	1	0	1
0	0	1	0	0	1	1
0	0	0	1	1	1	1

Distance d'un code linéaire

- $d = \text{Min} \{ d_H(x, y) ; \forall x,y \in C \}$
  $= \text{Min} \{ w_H(x - y) ; \forall x,y \in C \}$
  $= \text{Min} \{ w_H(z) ; \forall z \in C \}$

- Borne de Singleton : $d \leq n-k+1 = r+1$

- On peut donc corriger :
  - jusqu'à (d-1)/2 erreurs quelconques
  - Jusqu'à (d-1) erreurs localisées (effacements)

- Code MDS:
  - distance maximale: atteint la borne de Singleton: d = r+1

Décodage d'un code linéaire

- Un code linéaire est équivalent à un code de matrice génératrice $G = [Id_k \mid A]$

- *code*($[s_0, \ldots, s_{k-1}]$) = $s.G = [c_0, \ldots, c_{k-1}, c_k \ldots, c_{n-1}]$ où:
  $[c_0 \ldots, c_{k-1}] = [s_0, \ldots, s_{k-1}]$
  et $[c_k \ldots, c_{n-1}] = [s_0, \ldots, s_{k-1}].A = [c_0, \ldots, c_{k-1}].A$
  Intérêt: coût de codage : calcul de c=s.G => O(k.r) opérations

- De plus: $[c_k \ldots, c_{n-1}].Id_{n-k} - [c_0, \ldots, c_{k-1}].A = [0]_{n-k}$
- Soit (en transposant) $[-A^t \mid Id_{n-k}] \cdot c^t = 0$, i.e. $\mathbf{H} \cdot c^t = 0$

  On a aussi: $H.G^t = 0$

H = Matrice de contrôle de dimension (n-k, n)

- Rem: d = nbre minimal de cols indép. de H

Détection d'erreurs

- On émet x et on reçoit y;
  y est un mot de code $\Leftrightarrow H.y^t = 0$
- **Syndrome** d'erreur : $\mathbf{s = H .y^t}$
  $s \neq 0$ => il y eu erreur de transmission
- Exemple: matrice de contrôle du code de Hamming.

Correction d'erreurs

- $x = [x_1 .. x_n]$ émis ; $y = [y_1 .. y_n]$ reçu
- Corriger y $\Leftrightarrow$trouver x tq x=y- e appartient au code avec e = vecteur de correction (=erreur) de poids minimum
- Or: $s = H .y^t = H.x^t + H.e^t = H.e^t$
  Le syndrome donne toute l'information pour corriger :
  e = vecteur d'erreur de poids minimal tel que $H.e^t = s$
- Deux méthodes possibles :
  - Localisation des erreurs puis résolution du système linéaire $H.e^t = s$
  - Par tabulation de la correction e associé à chaque syndrome possible
    - on stocke dans un tableau Cor[s] := e pour chaque syndrome s
    - Ex: V= {0,1}, n=64 k=52 :
      On peut recevoir un mot parmi 2^{64} mots possibles, dont $2^{64} - 2^{52}$ mots erronés
      mais seulement 2^{12}=4096 syndromes possibles !
      donc table des corrections de taille 4096

Bons codes

- Facile et efficace à implémenter
  - Codage et décodage (détection/correction) peu coûteux
  - logiciel et/ou matériel
- Etant donné un taux de correction « t/n » donné, pouvoir facilement construire un code (n,k,d) qui permet ce taux de correction
- Exemples:
  - codes de Hamming
  - Codes cycliques

Code de Hamming

- Définition / Théorème de caractérisation :
  - Les codes de Hamming sont 1-correcteurs et 1-parfaits
  - Tout code binaire linéaire 1-correcteur parfait est un code de Hamming.

Code binaire 1-correcteur

- Code binaire de longueur n:
  - Si il y a 0 ou 1 erreur de bits => n+1 possibilités
- Au moins $r = \log_2 (n+1)$ bits pour coder une erreur possible
- Code de Hamming: code (2^r-1, k=2^r-r-1, d=3)
  $c_1 \ldots c_n$ défini par :
  - Si $i \neq 2^j$ alors c_i est un bit de source
  - Si $i = 2^j$ alors c_i est un bit de contrôle de parité = somme des c_k tel que k écrit en binaire a un 1 en position j
  - Exemple code de Hamming (7,4)
- Détection: si au moins un des bits de parité est erroné
- Correction: on change le bit d'indice la somme des indices des bits de parité erronés

Code de Hamming

- Code 1-correcteur à nombre de bits ajoutés minimal, δ = 3

🔔 Idée : ajouter un contrôle de parité pour chaque puissance de 2 ➔ b1, b2, b4, b8, b16, etc.

⇒ Cela suffit pour localiser l'erreur !

⇒ C(n, n-⌊$\log_2(n)$⌋-1) → rendement ≈ 1-$\log_2(n)/n$

$$\begin{matrix} & & & & b_1 & b_2 & & b_4 & & & \\ \end{matrix}$$

$$[x_0 \quad x_1 \quad x_2 \quad x_3] \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

⇒ rendements :
- 4/7 ≈ 57 %
- 11/15 ≈ 73 %
- 26/31 ≈ 84 %
- ...

Exemple : code de Hamming (7,4)

- C'est aussi un code linéaire sur V = {0,1}

engendré par G =

1	1	1	0	0	0	0
1	0	0	1	1	0	0
0	1	0	1	0	1	0
1	1	0	1	0	0	1

- Remarque: il est équivalent aux codes de matrices génératrices G'=L.G.P avec L inversible et P permutation. Par exemple:

G'=

1	0	0	0	1	1	0
0	1	0	0	1	0	1
0	0	1	0	0	1	1
0	0	0	1	1	1	1

G''=

1	0	1	0	1	0	0
0	1	0	0	1	0	1
0	0	1	0	0	1	1
0	0	0	1	1	1	1

Code du minitel

- Faible taux d'erreur, mais paquets longs :
  - Code de Hamming + detection paquets
- Source = suite de paquets de 15 octets = 120 bits
- Correction d'1 erreur : Code Hamming(127,120,3) :
  - 7 bits + 1 bit parité pour les 7 bits contrôle = 1 octet
- Détection de paquets: 1octet avec que des 0
  - Si erreur détectée : ARQ
- Total : 17 octets, rendement = 15/17 = 88%

III Codes cycliques

- Famille de codes linéaires avec distance garantie, faciles à construire et implémenter.
  - Cas particulier: codes de Reed-Solomon
- Très utilisés dans les applications pratiques

Codes cycliques

- **Rappel**: C code linéaire (n,k) == code engendré par une matrice G (k lignes, n colonnes) de rang k;
  les lignes de G sont formées par k mots de code linéairement indépendants.
- **Déf 1: Opérateur décalage**: $\sigma([c_0 \ldots, c_{n-1}]) = [c_{n-1}, c_0 \ldots, c_{n-2}]$
- **Déf 2** : Code cyclique $\Leftrightarrow$ code linéaire stable par σ
- Exemple: **code binaire cyclique (7,4) qui contient « 1011000 »**
- **Théorème 1: matrice génératrice d'un code cyclique**

  Un code cyclique (n,k) est équivalent à un code engendré par un mot de code
  $m=[c_0 , \ldots, c_{n-k-1} , c_{n-k}=1, 0 \ldots, 0]$
- Intérêt 1: description simple, seulement r=n-k symboles de V

Caractérisation: polynôme générateur

- Tout mot $u \in V^n$ peut être représenté par un polynôme de degré n-1 :
$$u = [u_0, \ldots, u_{n-1}] \Leftrightarrow P_u = \Sigma_{i=0}^{n-1} u_i.X^i$$

- **Lemme** : le mot $\sigma(c)$ est associé au polynôme :
$$P_{\sigma(C)} = X.P_c \bmod (X^n - 1)$$

- **Définition 3**: **polynôme générateur** du code cyclique généré par le mot de code $m=[c_0, \ldots, c_{r-2}, c_{r-1}, c_r=1, 0 \ldots, 0]$
$$\mathbf{g(X) = P_m = \Sigma_{i=0}^{r-1} c_i.X^i + X^r}$$

- **Théorème 2** : $\forall$ c mot de code, P_c est multiple de g(X)

- **Théorème 3** : g est un diviseur unitaire de $X^n - 1$ de degré r

Codage/décodage code cyclique

- **Codage** : $P_{[a.G]} = g(X).P_a$
  - Tout mot de code est un multiple de g mod $X^n - 1$
  - Tout multiple de g mod $X^n - 1$ est un mot de code

- **Détection** : on reçoit $y = P_y$
  - Si P_y n'est pas multiple de g => erreur
  - Syndrome d'erreur: $P_e = P_y \bmod g$

- **Correction**: à partir du syndrome
  - (algorithme de Meggitt)
  - Si Poids (P_e) inférieur à (d-1)/2 : correction : $P_y - (P_y \bmod g)$
  - Cas général: algorithme de Berlekamp-Massey en O(n log n)

Distance minimale d'un code cyclique

- **Théorème 4 (dit BCH)** (n premier avec q)
  - Soit α racine primitive de $X^n - 1$ dans GF(q)
  - Si il existe a et b entiers tel que g(X) est multiple de
    $(X- \alpha^a)\ (X- \alpha^{a+1})\ (X- \alpha^{a+2})\ \ldots\ (X- \alpha^{a+b-1})$

Alors le code cyclique C de polynôme générateur g(X) est de distance : **d(C) ≥ b+1**

C est donc au moins [b/2]-correcteur (ou b-détecteur)

Codes cycliques

- Code cyclique : $u=[u_1,\ldots,u_n] \in C \Leftrightarrow \sigma(u)=[u_n,u_1\ldots,u_{n-1}] \in C$
- Exemple : le code de parité (n,n-1);

- Tout code cyclique admet une matrice génératrice de la forme :
  - $[\ m\ ,\ \sigma(m),\ \ldots\ ,\ \sigma^{k-1}(u)\ ]^T$ avec $m=[a_1,\ldots,a_{n-k}=1,0,\ldots,0]$
  - Et $g(X) = \sum a_i X^i$ (polynôme générateur) est un diviseur de (X^n-1)

  - Codage : $P_y = g \times P_u \bmod (X^n-1)$ (temps soft linéaire FFT)
  - Décodage : $P_u = P_v / g$
  - Correction :
    - Tabulation du syndrôme d'erreur $P_e = P_v \bmod g$
    - Méthode de Meggitt ; etc.

Bose-Chaudhuri-Hocquenghem, Reed-Solomon

- BCH
  - Racines de g ¾ $\{\alpha^{i+1},\ldots,\alpha^{i+2t}\}$ ➔ δ ¸ 2t+1 !
  - Intérêt : correction par Berlekamp-Massey, quadratique sans table !

- Reed-Solomon
  - BCH avec V = F(2^m) et n=q-1=2^m-1
  - Nombre de chiffres de redondance minimal pour une correction donnée : (n,k=n-r,r+1) avec r le degré du polynôme générateur.
    - Exemple : NASA pour les communications spatiales
      - $g = \prod_{j=12}^{43} (X-\alpha^{11j})$ sur F(2^8)
      - Code (255,223,33)

Codes binaires de Reed-Solomon

- **On choisit : $q = 2^m$** (donc V= {chiffres de m bits}) et **$n = 2^m - 1$**
- On a alors: $X^n - 1 = X^{q-1} - 1 = \prod_{a\in GF(q)^*} (X - a)$.
  - Si α générateur de GF(q)* : { a ∈ GF(q)* }={α^i / i=0..q-2}
    Donc $X^n - 1 = \prod_{a\in GF(q)^*} (X - \alpha^i)$
  - D'après BCH, il suffit de choisir $g = \prod_{i=a..a+r-1} (X - \alpha^i)$!!!

- **Définition 4** : code cyclique de Reed-Solomon RS(n, k) :
  RS(n,k) est un code cyclique de polynôme générateur
  $g(X) = \prod_{i=a..a+r-1} (X - \alpha^i)$ de degré r

- **Théorème 5** : RS(n,k) est de distance d = n-k+1 = r+1
  - D'après théorème BCH : d ≥ r+1
  - D'après la borne de Singleton: d ≤ r+1

- Ex. Galileo : RS(255,223) sur V={octets}, 16-correcteur
- NB RS(n,k) est de distance maximale sur GF(2^m) !!!
  … mais pas nécessairement parmi les codes binaires…

Dimensionnement d'un code binaire RS

- *Canal binaire symétrique* (BSC) avec un taux τ d'erreurs de bits. Exemple: τ=1%
  Définition: C = Capacité BSC(τ) = 1+ τ.log2 τ + (1- τ).log2(1- τ)

  On suppose m fixé: V=GF(2^m); un chiffre de V = m bits. (parfois m=8, 16, 32 etc)
  p=Prob (erreur transmission d'un chiffre de m bits) = 1-(1- τ)^m.
- Un code RS(n,k) sur V est de rendement k/n et corrige t=(n-k)/2 erreurs de chiffres.
  La probabilité ε d'une erreur non corrigée (***erreur résiduelle***) est :

$$\varepsilon \leq \sum_{i=t+1}^{n} C_n^i p^i (1-p)^{n-i} = \sum_{j=0}^{n-t-1} C_n^j p^{n-j} (1-p)^j$$

- **Théorème de Shannon** : pour tout R < C, il existe un code de rendement R et d'erreur résiduelle arbitrairement petite.
- Exemple: τ=1% => capacité = 0,91… => on cherche un code correcteur de rendement proche de 91% et d'erreur résiduelle faible.

Le choix de m et t définit le code RS … et impose R et ε

- Pour **τ=5.10^{-4} = 0,0005 :** rendement < capacité = 0,993

m	t	r	n	R	ε
8	2	4	255	0,984	0,083314104
8	4	8	255	0,969	0,003850848
8	8	16	255	0,937	1,16402E-06
8	12	24	255	0,906	6,0687E-11
8	16	32	255	0,875	9,02514E-16
8	32	64	255	0,749	1,01325E-38
16	500	1000	65535	0,985	0,830965135
16	600	1200	65535	0,982	0,00038756
16	700	1400	65535	0,979	4,80635E-14
16	800	1600	65535	0,976	4,27197E-30
16	1000	2000	65535	0,969	5,75845E-78

=> Code(255, 223) sur GF(256)
de rendement 87% et 16-correcteur
avec erreur résiduelle = 10^{-15}

NB Chiffres de 8 bits:
bloc de 1784 bits codés par 2040 bits

- Pour **τ=1% = 0,01 :** rendement < capacité = 0,91

m	t	r	n	R	ε
8	8	16	255	0,9373	0,998133479
8	16	32	255	0,8745	0,769179312
8	32	64	255	0,749	0,002590694
8	48	96	255	0,6235	3,08393E-09
8	64	128	255	0,498	6,64789E-18
8	80	160	255	0,3725	7,67393E-29
16	9000	18000	65535	0,7253	1
16	10000	20000	65535	0,6948	0,001808289
16	10500	21000	65535	0,6796	4,22996E-17
16	11000	22000	65535	0,6643	7,73789E-43

Exemples classiques

- Minitel(136,120) = Hamming(128,120) 1-correcteur avec un ajout de 8 bits toujours à 0, pour les grosses erreurs.
  - » taux d'erreur = 10^{-4}; rendement ≈ 88 %; bits ajoutés = 16
- Consultative Commitee for Space Data Standard : échange de données spatiales avec RS(255,223)

  ⇒ 32-détecteur et 16-correcteur; rendement ≈ 87,5 %; bits ajoutés = 256
- CD audio : CIRC(32,28) (Cross Interleaved RS Code)
  1. RS(255,251) → distance 5
  2. On ne prend que les mots de code commençant par un nombre donné de 0, puis on enlève les 0 → (32,28) = RS raccourci.
  3. La distance est conservée, donc aussi 4-détecteur et 2-correcteur
     - » taux d'erreur = 10^{-5}; rendement ≈ 87,5 %; bits ajoutés = 32

Codes

V : un corps (fini) => un mot de V^k est vu comme un polynome de degré k-1 à coefficients dans V.

1. Code par interpolation codage
2. Exemple simple
3. Décodage
4. Reed-Solomon: Codage et décodage par FFT
5. Extension au cas entier

Evaluation / Interpolation

- M=[a_0, …, a_{k-1}] mot source
- $P_m(X) = \Sigma_{i=0..k-1}\ a_i.X^i$ de degré k-1 sur V
  - caractérisé de manière unique par sa valeur en k points x_i distincts

- Pour **n ≥k** (et n ≤ #V), soient n points ≠ x_j et $y_j = P_m(x_j)$

  Alors **$[a_0, …, a_{k-1}] \Leftrightarrow [y_0, …, y_{n-1}]$**
- Dém:
  - $(a_j) => (y_j)$: **évaluation** Pour j=0, … n-1: $y_j = \Sigma_{i=0..k-1} a_i.x_j^i$
  - $(y_j) => (a_j)$: **interpolation**: $\Sigma_{i=0..k-1} a_i.X^i = \Sigma_{i=0..n-1}\ y_i\ L_i.(X)$

    avec $L_i.(X) = \Pi_{j\neq i}\ (x_i - x_j)^{-1}.(X - x_j)$: polynôme de Lagrange

Matrice génératrice du code

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \dots & 1 \\ x_1^1 & x_2^1 & \square & \square & \dots & \square & \square & \square & \square & x_n^1 \\ x_1^2 & x_2^2 & \square & \square & \square & \square & \square & \square & \square & x_n^2 \\ . & \square & \square & \square & . & \square & \square & \square & \square & . \\ x_1^{k-1} & x_2^{k-1} & \square & \square & \dots & \square & \square & \square & \square & x_n^{k-1} \end{pmatrix}$$

Interpolation et correction

- Théorème fondamental:
  Soient n évaluations (y_j) d'un polynôme P de degré k-1
  Parmi ces n évauations, t sont erronées.

  Si $2t \leq n-k$, alors P est l'unique polynôme de degré $< k$ qui corresponde à au moins n-t évaluations correctes, i.e.

  $$\#\{ j=1..n : y_j \neq P(x_j) \} \leq t$$

Correction

- Soit Q(X) le polynôme qui interpole les (y_j) en les n points:
  $$Q(X) = P(X) + E(X) \text{ avec E = polynôme d'erreur}$$

- Soit $\Pi(X)=\Pi_{j=0..n-1}(X - x_j)$
- Soit I le sous ensemble des n-t points x_j tels que $P(x_j) = y_j$ [corrects]

- Pour tout x_j de I : $E(x_j)=0$
- Donc E(X) est un multiple de $\Pi_V(X)= \Pi_{j \text{ dans } I}(X - x_j)$:
  $$E(X) = Z(X).\ \Pi_V(X)$$
  et PGCD (E(X) , Π(X)) = $\Pi_V(X)$ donc de degré $n-t \geq k+t$

- Astuce: les premières étapes du calcul PGCD(Q(X)=P+E, Π(X)) donnent les mêmes quotients que PGCD(E(X), Π(X)) !!!

Algorithme de correction par Euclide tronqué ["reconstruction rationnelle" / Berlekamp-Massey]

- Entrée: Π , Q = polynôme d'interpolation aux n points y
- Sortie: P de degré ≤ k-1 qui correspond à n-t évaluations

- $A_0=1.X^0$; $B_0 = 0$; $R_0=\Pi$;
  $A_1=0$; $B_1=1.X^0$; $R_1=Q$;
- For (i=1 ; $deg(R_i) \geq n - t$; i+=1)
  - Soit q_i le quotient euclidien de R_{i-1} par R_i.
  - $R_{i+1} = R_{i-1} - q_i.R_i$;
  - $A_{i+1} = A_{i-1} - q_i.A_i$;
  - $B_{i+1} = B_{i-1} - q_i.B_i$;
- /If (Ri mod Bi ≠ 0) { error ("**Trop d'erreurs: correction impossible**") ; }
  else return $P = R_i / B_i$; // B_i est un multiple de R_i,

Codage et Décodage en Maple

```
##### Fonction de CODAGE d'un mot a
#####  (i.e. evaluation du polynome Pa(X) aux abscisses x[i] i=1..n
CodageInterpol := proc( a::list)::list ;
  local Pa, y, i ; # Pa est le polynome associé à a; y le vecteur des évaluations
  description "Codage par interpolation aux n points x[i] du mot source a de longueur k »;
  Pa := sum( op(i+1,a)*'X'^i, i=0.. k-1) mod q :
  y := [seq( eval(Pa, X=x[i]) mod q, i=1..n )] :
  print("Polynome associé au mot source:", Pa) :
  print("Mot de code transmis :", y) :
  y
end proc;
```

Polynôme d'interpolation (formule de Lagrange)

Algorithme d'Euclide tronqué (reconstruction rationnelle)

Polynôme source après correction

```
##### Fonction de DECODAGE ET CORRECTION par algorithme d'Euclide tronqué
DecodageInterpol := proc( yrecu::list)::list ;
  local
    Precu,  # Le polynome d'interpolation de degré n associé au mot recu
    Pcorr, motcorrige, # Le polynome corrigé et le mot associé
    A0, B0, A1, B1, # Les coefficients de Bezout dans Euclide
    i, aux ;# Variables internes pour les boucles et  pour les permutations
  description "Decodage par interpolation aux n points x[i] du mot recu de longueur n";
  Precu:= expand( sum(  op(i,yrecu)*op(i,LL), i=1..n )) mod q;
  print("Mot reçu : yrecu = ", yrecu ) :
  print("Interpolation de yrecu  :", Precu ) :
  A0:=1: B0:=0: R0 := PI:  print("Reste Euclide tronqué:",  R0) :
  A1:=0: B1:=1: R1 := Precu: print("Reste Euclide tronqué:",  R1):
  while (degree(R1) >= n-t) do
    quotient := Quo(R0, R1, X) mod q:
    aux := Expand(R0 - quotient*R1) mod q: R0:=R1: R1:=aux : print("Reste Euclide tronqué:",  R1):
    aux := Expand(A0 - quotient*A1) mod q: A0:=A1: A1:=aux :
    aux := Expand(B0 - quotient*B1) mod q: B0:=B1: B1:=aux :
  end do :
  if ( Rem(R1,B1,X) mod q <> 0) then
    print ("*** ERREUR: CORRECTION IMPOSSIBLE *** Reste non nul=", Rem(R1, B1, X) mod q  ) :
    motcorrige :=[ "ECHEC DECODAGE car trop d'erreurs!" ]
  else
    Pcorr := Quo(R1, B1, X) mod q:
    motcorrige := [seq(coeff(Pcorr,'X',i), i=0..max(degree(Pcorr),k-1) ) ] :
    print( "Polynome interpolé corrigé=", Pcorr, " qui correspond au mot source:", motcorrige)
  end if :
  motcorrige
end proc;
```

Un exemple dans GF(11)

- Evaluation aux abscisses (0, 1, 2, 3, 4, 5) donc Code (6, 2, 5) qui est t=2 correcteur

Codage Mot source: **[7,2]**
Polynôme associé au mot source: $P = 7 + 2X$
Mot de code transmis : **[7, 9, 0, 2, 4, 6]** (i.e. évaluation de P aux absisses X= 0, 1, 2, 3, 4, 5)

Décodage: Mot reçu avec 2 erreurs: **[7, 4, 0, 2, 6, 6]**
1/ Polynôme d'interpolation aux valeurs reçues: $Q := 2X^5 + 2X^4 + 7X^3 + 6X^2 + 2X + 7$

2/ Séquence des restes dans l'algorithme d'Euclide étendu (et coefficient de Bezout B associé):
Étape 0: $R_0 = L(X) = X(X-1)(X-2)(X-3)(X-4)(X-5)$ $[B_0 = 0]$
Étape 1: $R_1 = Q(X) = 2X^5 + 2X^4 + 7X^3 + 6X^2 + 2X + 7$ $[B_1 = 1]$
Étape 2: $R_2 = R_0 \bmod R_1 = 4X^4 + 4X^3 + 2X^2 + 8X + 1$ $[B_2 = 5X + 8]$
Étape 3: $R_3 = R_1 \bmod R_2 = 6X^3 + 2X^2 + 7X + 7$ $[B_3 = 3X^2 + 7X + 1]$
Arrêt de l'algorithme d'Euclide car degré inférieur ou égal à n-t = 4.

3/ Correction: Calcul de $R_3/B_3 = (6X^3 + 2X^2 + 7X + 7) / (3X^2 + 7X + 1) \bmod 11 = \mathbf{7X + 2}$
=> Mot décodé: **[7, 2]**

Lien avec Reed Solomon

- Produit matrice-vecteur rapide => FFT
  - Choix pour x_i = racines de l'unité
  - Dans GF(Q) => q-1 racines de l'unité distinctes = ω^i
  - Donc limite le degré $n \leq q-1$
- Reed-Solomon: code d'interpolation de rendement maximal avec les racines de l'unité comme abscices :
  - $n = q-1$
  - $k \leq n$ arbitraire. Permet de corriger $t = (n-k)/2$ erreurs.
  - Codage: en $O(n \log n)$ [FFT]
  - Décodage en $O(n \log^2 n)$ [Euclide rapide ou Berlekamp-Massey]
- Exemple: Consultative Commitee for Space Data Standard : échange de données spatiales avec RS(255,223)
  ⇒ 32-détecteur et 16-correcteur; rendement ≈ 87,5 %; bits ajoutés = 256

Construction du code et test (en Maple)

```
##### CARACTERISTIQUES DU CODE - Constantes globales
q := 11 :  # Le corps de base : Z/11Z
n:=6 : k:=2 : t := (n-k)/2 :
print(cat("Code Interpolation ( ", n, ", ", k, ", ", n-k+1, " ) sur GF(", q, ")qui est ", t,"-correcteur.")):

X := 'X' ; # X est l'indeterminee pour les polynomes a 1 variable.
x:= [seq(i mod q,i=0..n-1)]: print("Abscisses d'evaluation: ", x) :  # Les points d"evaluation
# La base de Lagrange: LL(i), i=1..n
PI:= product( (X-op(i,x)), i=1..n ) mod q ;
LL:=[seq( (expand(PI/(X-op(i,x)mod q)) mod q) * (eval(expand(PI/(X-op(i,x)mod q))mod q, X=op(i,x))^(-1) mod q), i=1..n )] mod q:


#### Fonction de GENERATEUR ALEATOIRE D'ERREUR DANS UN MOT DE TAILLE n
GenereErreurCanal := proc( mot::list, nberreurs::integer)::list ;
  local   err,  # err est le vecteur d'erreur généré aléatoirement et de poids = nberreurs
          poserreurs, # l'ensemble de toutes les positions des erreurs
          pos, j, tmp ; # variable locale
  description "Renvoie (mot + err), mot de longueur n, où err est un vecteur d'erreur de poids de Hamming=nberreurs";
  err := [seq( 0, i=1..n)]: # Pour generer un vecteur d'erreur aleatoire initialisé à 0 erreurs
  poserreurs := {} :
  while ( nops(poserreurs) < nberreurs)  do
    pos := RandomTools[Generate](integer(range=1..n)) :
    tmp := poserreurs union {pos}  : poserreurs := tmp ;
  end do:
  for j in poserreurs do err[j] := RandomTools[Generate](integer(range=1..q-1)) end do:
  (mot + err ) mod q
end proc;

TestCodageDecodage := proc( source::list, nberreurs::integer)::list :
  local     motcode, motrecu, motcorrige:
  motcode := CodageInterpol( source ) :
  motrecu := GenereErreurCanal( motcode, nberreurs ) :
  motcorrige := DecodageInterpol( motrecu ) :
  print("Mot source=", source, " -- Mot transmis=", motcode, " -- Mot recu=", motrecu, " -- Apres correction=", motcorrige):
  motcorrige
end proc
```

Exemple:
TestCodageDecodage([7,2], 2) ;

```
#Reed Solomon: abscisses = racines de l'unité
n := q-1;
k := n-4;
alpha := 7; # [seq( 7**i mod 11, i=1..10)]= [7, 5, 2, 3, 10, 4, 6, 9, 8, 1]
x:= [setCodageDecodage(alpha^i mod q,i=0..n-1)]:
print("Abscisses d'evaluation: ", x) :  # Les points d"evaluation
# La base de Lagrange pour ReedSolomon: LL(i), i=1..n
PI:= product( (X-op(i,x)), i=1..n ) mod q ;
LL:=[seq( (expand(PI/(X-op(i,x)mod q)) mod q)
     * (eval(expand(PI/(X-op(i,x)mod q))mod q, X=op(i,x))^(-1) mod q), i=1..n )] mod q:
```


Reed-Solomon et Codes cycliques

- Pour Reed Solomon, on pourrait prendre 0 en plus des racines de l'unité => Code MDS (n, k, n-k+1) sur V de card $n=p^m$

☺ Si on ne prend que les n-1 racines de l'unité, alors si P(X) est un polynôme de degré k-1, $[\ P(\omega^i)\ ]_{i=0..n-2}$ est un mot de code.
Soit Q(X) = P(ωX) : $[\ Q(\omega^i)\ ]_{i=0..n-2}$ = shift ($[\ P(\omega^i)\ ]_{i=0..n-2}$).
Tout décalage circulaire d'un mot de code est un mot de code!

☺ Plus généralement, il est possible de construire des codes linéaires avec une distance minimale donnée : les codes cycliques.

- Exemple les codes de Reed-Solomon sont cycliques, mais imposent n=q-1.
- Ex: les x_i sont des octets (m=8), il y en a 2^8-1=255
  ⇒ RS(255,253) est alors 2-détecteur et 1-correcteur pour un ajout de 2 octets ($16=\lfloor \log_2(2^8 x 255) \rfloor +1$ bits ➔ c'est Hamming(2040,2024))

Généralisation: anneau euclidien

- Remplacer degré(M) par log(M) [attention: log *réel*]

Évaluer P en a $\leftrightarrow$ Reduire P modulo $X - a$

Polynômes	**Entiers**
Evaluation: $P \mod X - a$ Évaluer P en a	$N \mod m$ "Évaluer" N en m
Interpolation: $P = \sum_{i=1}^{k} \frac{\prod_{j \neq i}(X - a_j)}{\prod_{j \neq i}(a_i - a_j)}$	$N = \sum_{i=1}^{k} a_i \prod_{j \neq i} m_j (\prod_{j \neq i} m_j)^{-1[m_i]}$

IV Codes binaires de Reed Solomon et dimensionnement

IV Codes de Reed Solomon
-
Définition par interpolation

Codes de Reed-Solomon

- **Théorème**: Soient F un corps de cardinal $\geq n$ et $\alpha_1,\ldots,\alpha_n$ n éléments $\neq$ de F.
  Pour $1 \leq k \leq n$, $RS(n,k) = \{ [Q(\alpha_1),\ldots,Q(\alpha_n)] : Q(X) \in F[X], \deg(Q) \leq k \}$
  est un code de Reed-Solomon (n,k) sur F.
  Ce code est de dimension k+1 et de distance n-k (maximale):

  Code (n, k+1, n-k) sur F

- Problème des codes de Reed Solomon: $|V| \geq n$, donc $q=|V|$ grand
  - Pour un bloc de b bits: donc choisir $V=F(2^m)$ et n tels que : $(m \geq \log_2 n)$ et $(m.n \geq b)$.
    - Exemple: canal binaire, correction d'un taux d'erreur de 2% :
      - » b=100 => code (20, 16, 5) sur F(32) Rendement = 16/20 = 80%
      - » b=1000 => code (125, 85, 41) sur F(256) Rendement = 85/125 = 68%
      - » b=10000 => code(1000, 600, 401) sur F(1024) Rendement = 600/1000 = 60%

Décodage unique de Reed-Solomon par Berlekamp-Welch

- **Problème décodage avec e≤(d-1)/2=(n-k)/2 erreurs et t=n-e≥k+e+1 valeurs correctes.**
  - Entrée: $y_1, \ldots, y_n \in F^n$ vérifiant $\exists P \in F[X]$ de degré k tel que $\#\{i / y_i = P(\alpha_i)\} \geq t$.
  - Sortie : le polynôme P unique tel que $\#\{i / y_i = P(\alpha_i)\} \geq t$.
    - *Preuve unicité*: Si P_1 et P_2 sont solutions: P_1-P_2 s'annulle en au moins n-2e ≥k+1valeurs. Or deg(P_1-P_2)≤k, d'où P_1=P_2.

- **Principe** :Soit $T=\{ i / y_i = P(\alpha_i) = L(\alpha_i) \}$; et $E(X)= \Pi_{i \notin T} (X-\alpha_i)$ le polynôme localisateur d'erreurs.
  - Soit L(X) le polynôme d'interpolation aux n points (α_i, y_i); degré(L(X)) ≤ n-1.
    Degré(E(X))≤e et pour i=1..n : $E(\alpha_i).L(\alpha_i) = P(\alpha_i).E(\alpha_i)$.
    Posons N(X)=P(X).E(X) qui est de degré k+e: on a donc pour i=1..n : $E(\alpha_i).L(\alpha_i) = N(\alpha_i)$
    ie un système de n équations à k+2.e+1≤n inconnues (k coefs de E et k+e+1 de N);
    comme P est unique, ce système admet une solution unique.
- **Algorithme**
  - Étape 1: calculer le polynôme d'interpolation L(X) et former le système linéaire: $y_i.E(\alpha_i) - N(\alpha_i) = 0$ i=1..n
  - Étape 2: résoudre le système linéaire: on obtient les coefficients de E(X) et N(X).
  - Étape 3: retourner le polynôme N(X) / P(X).

- **Remarque**: il existe des algorithmes plus rapides en $O(n.\log^2 n)$.
  [Berlekamp-Massey, ou calcul de pgcd tronqué en utilisant un algorithme de pgcd rapide].

Codes concaténés

- Code concaténé: construit sur V avec q=|V| petit à partir de 2 codes:
  - « inner code » : C_{in} (n_{in}, k_{in}, d_{in}) avec Q mots sur le petit alphabet (V_{in} de taille q)
  - « outer code » : C_{out} (n_{out}, k_{out}, d_{out}) sur un grand alphabet (V_{out} de taille Q)

Code concaténé série

- **Codage:**

Mot source

k_{out} symboles de V_{out}, soit $k_{out}.k_{in}$ symboles de V_{in}

(codage C_{out})

⇓

n_{out} symboles de V_{out}, soit $n_{out}.k_{in}$ symboles de V_{in}

(codage C_{in})

⇓

Mot de code

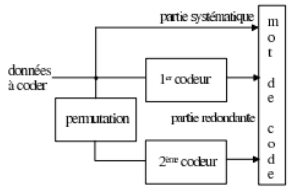
$n_{out}.n_{in}$ symboles de V_{in}

Donc: Code($n_{out}.n_{in}$, $k_{out}.k_{in}$, $d_{out}.d_{in}$) sur V_{in} , de rendement $R_{out}.R_{in}$.

Autres concaténations possibles

(avec 2 codes ou plus)

- Concaténation parallèle (turbo-code)



Rendement = $R_1.R_2 / (1-(1-R_1)(1-R_2)) > R_1R_2$

- Code produit, code croisé

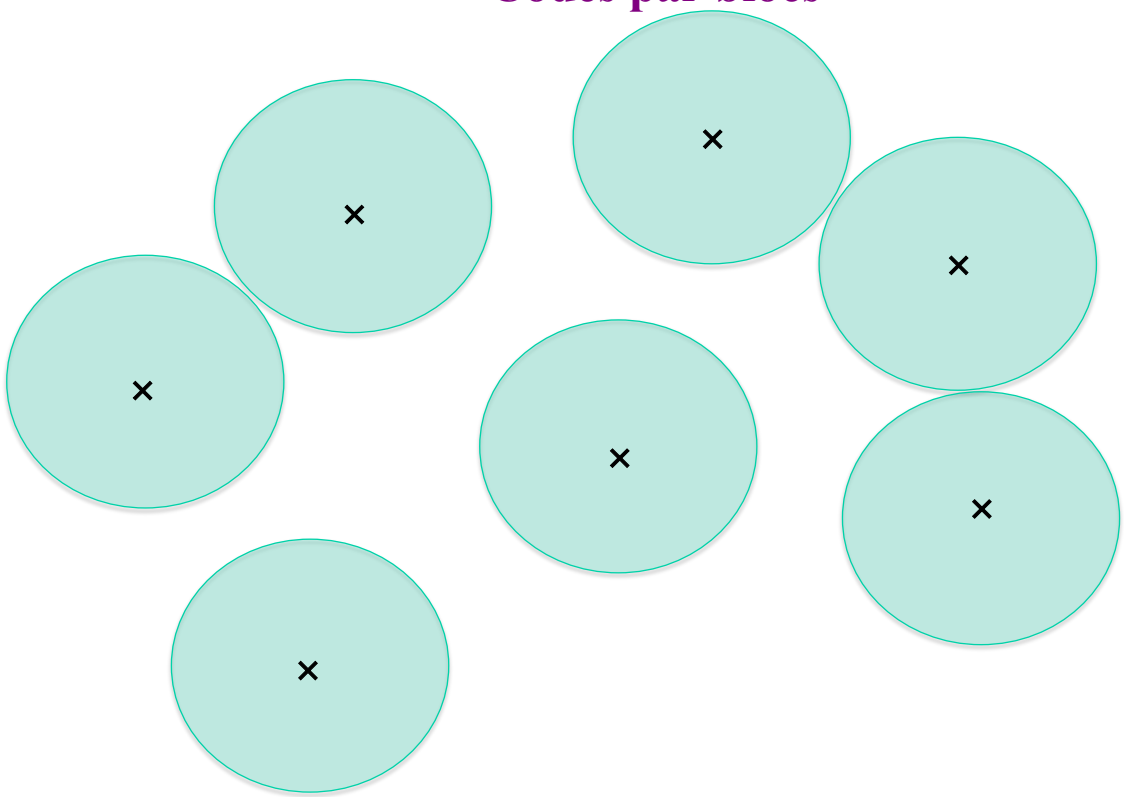
- Concaténation mixte série et parallèle

Chapitre. Décodage en liste (List Decoding)

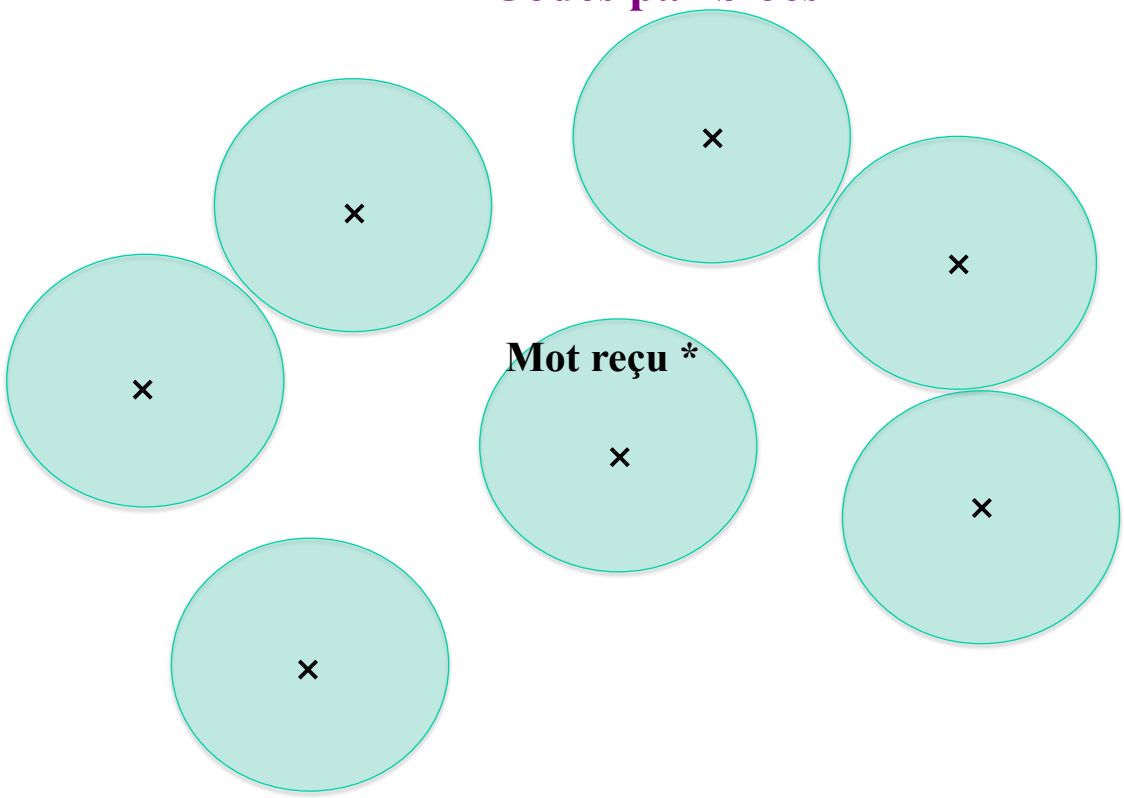
- Définitions: Décodage en liste et recouvrement en liste
- Bornes de Johnson
- Existence de codes linéaires décodables en liste
- Un algorithme de décodage en liste

- Référence:
  - Venkatesan Guruswami, “Algorithmic results in list decoding”, in Fundations and trends in theoretical computer science vol. 2, n.2, 2006, pages 107-195.

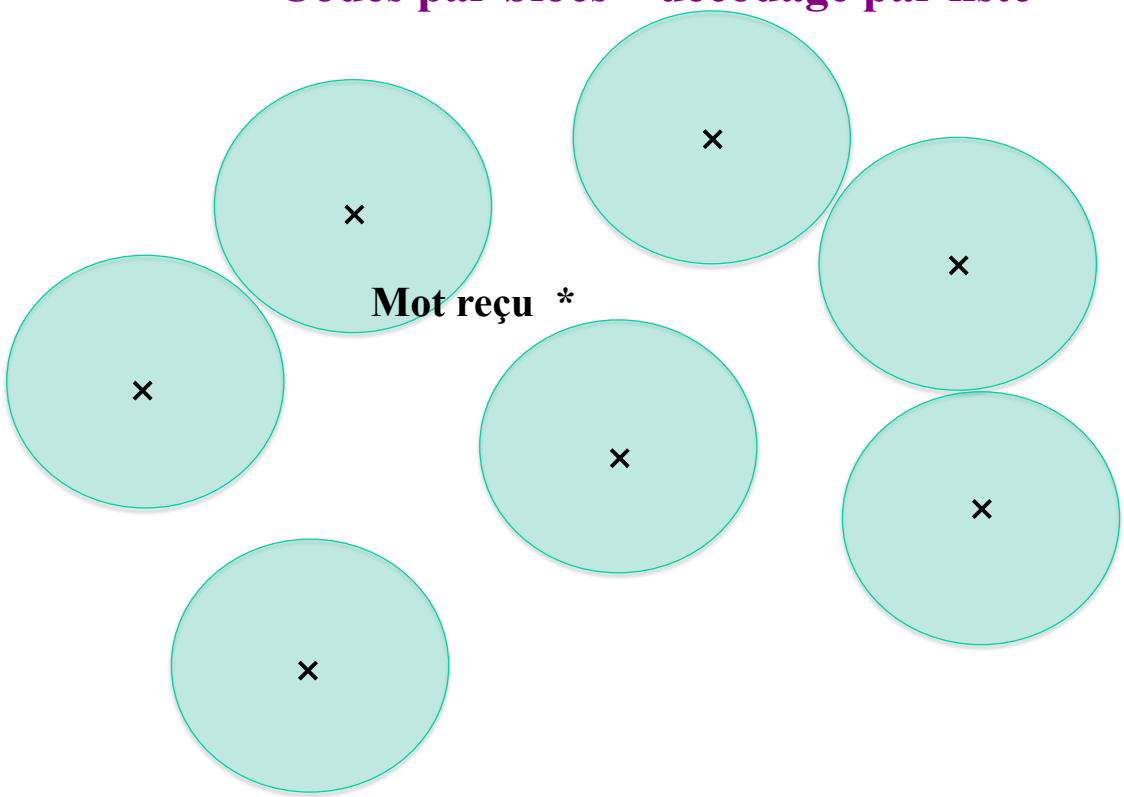
Codes par blocs



Codes par blocs



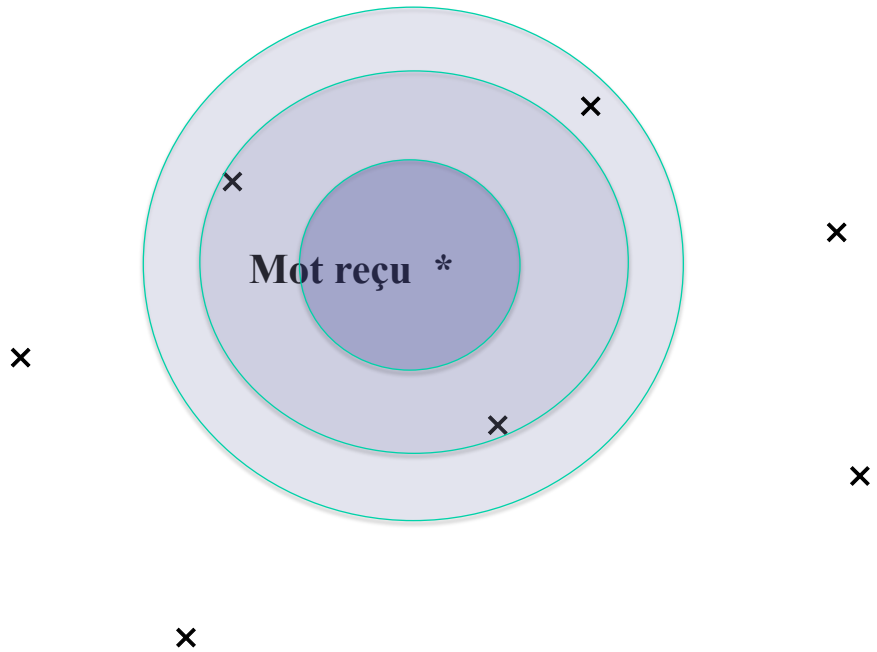
Codes par blocs – décodage par liste



Codes de Reed-Solomon et FFT

- Codage = évaluation d'un polynôme en n = k+r points (FFT)
  - Redondance de r symboles
  - Mot de code $[y_0, \ldots, x_{k-1}, y_k, \ldots, y_{n-1}] = \Omega\,[x_0, \ldots, x_{k-1}, 0, \ldots, 0]$
- Distance « maximale » (au sens classique)
  - Avec 2r symboles de redondance, on corrige r erreurs
- Correction: on reçoit $z=[z_0, \ldots, z_{k-1}, z_k, \ldots, z_{n-1}] = y+e$ avec e de poids $\leq r$
  - $\Omega^{-1}z = \Omega^{-1}y + \Omega^{-1}e = x + \hat{e} = [x_0+\hat{e}_0, \ldots, x_{k-1}+\hat{e}_{k-1}, \mathbf{\hat{e}_k, \ldots, \hat{e}_{n-1}}]$
  - les *r* symboles de $\mathbf{\hat{e}_k, \ldots, \hat{e}_{n-1}}$ engendrent linéairement $\hat{e}$ donc de calculer e

Codes par blocs – décodage par liste



Décodage en liste (List-decoding)

- **Problème du décodage en liste** d'un code C (n,k) sur V jusqu'à une fraction d'erreur p (ou rayon p.n) :
  - entrée : $y \in V^n$
  - sortie : la liste des mots de C à distance $\leq$ p.n de x, ie $\{ c \in C : d_H(c,y) \leq p.n \}$

- **Définition**: **(p,L) – décodabilité en liste :**
  pour $0<p<1$ et un entier L, un code $C \subset V^n$ est dit **(p,L)-décodable**
  (ou décodable jusqu'à une faction p d'erreurs avec une liste de taille L)
  ssi pour tout $y \in V^n$: $\text{Card}\{ c \in C : d_H(c,y) \leq p.n \} \leq L$.

  - La famille de codes $(C_n)_{n \in N}$ est (p,λ)-décodable ssi $\forall n$: C_n est (p, λ(n))-décodable avec $\lambda: N^* \rightarrow N^*$.
    - Cas particulier: si λ est constante égale à L, la famille est dite (p-L)-décodable.

Recouvrement en liste (List-recovering)

- **Définition**: (list-recovering): pour $0<p<1$ et un entier $u \leq L$,
  un code $C \subset V^n$ est dit ***(p, u, L)*-recouvrable** ssi:
  pour toute séquence $S_1, S_2, \ldots, S_n$ de n sous-ensembles de V avec $|S_i| \leq u$,
  $\text{Card}\{ c=[c_1, \ldots, c_n] \in C$ tels que « $\#\{i: c_i \in S_i\} \geq (1-p).n$ » $\} \leq L$.
  - La valeur « *u* » est appelée *taille de la liste d'entrée*.

- Remarque 1: (p, L)-décodable == (p, 1, L)-recouvrable.

- Remarque 2: (0,u,L)-recouvrement de liste dans le cas non bruité (p=0):
  - Entrée: u possibilités pour c_1, u possibilités pour c_2, …, u possibilités pour c_n
  - Sortie: les au plus L mots de code correspondant à ces possibilités.

- **Théorème**: Le code concaténé-série d'un *outer code* (p_1, u, L)-recouvrable et d'un *inner code* (p_2, u)-décodable est ($p_1.p_2$, L)-décodable en liste.

Borne de Johnson et existence de codes (p,L) décodables en liste

- **Notations: C** code (n, k, d) sur V avec |V|=q
  - $\mathbf{B_q(c, r)}$ [ou **B(c, r)**]: boule de Hamming de V^n centre $c \in V^n$ et de rayon r).
  - **B**=$C \cap B_q(c, r)$ et **(B 2)** = tous les sous-ensembles de cardinal 2 de B.
- **Quel est le nombre de mots de code dans $\mathbf{B_q(x, r)}$?**
  - Si $r \leq (d-1)/2$: $|B| \leq 1$, donc le rayon de décodage pour une liste de taille 1 est (d-1)/2.
  - **Cas général**: *Borne de Johnson*: soient $a=E_{x \in B}[\, d_H(c,x)]$ et $b=E_{\{x,y\} \in (B\ 2)}[\, d_H(x,y)]$, et soit $D= (1- a.q.(n.q - n)^{-1})^2 - (1 - b.q.(n.q - n)^{-1})$ alors si D>0:
    $$|B| \leq (1 - b.q.(n.q - n)^{-1}) / D$$
- **Corollaire** :
  - soient $\delta \in]0,1[$ tel que $d = n.(1 - q^{-1})(1-\delta)$ et $\gamma \in]0,1[$ et $r = n.(1 - q^{-1})(1-\gamma)$: si $\gamma^2 > \delta$, alors pour tout $c \in V^n$: $| B_q(c, r) \cap C | \leq (1 - \delta) / (\gamma^2 - \delta)$
  - *Preuve*: soient δ' et γ' tels que $E_{x \in B}[\, d_H(c,x)]=n.(1 - q^{-1})(1-\gamma')$ et $E_{\{x,y\} \in (B\ 2)}[\, d_H(x,y)]= n.(1 - q^{-1})(1-\delta')$. La borne de Johnson s'écrit: $|B| \leq \delta' (\gamma'^2 - \delta')$. Or $\delta' \leq \delta < \gamma^2 \leq \gamma'^2$, d'où le corollaire.
- **Cas particulier:** si q grand et si $r \leq n-(n^2-nd+1)^{1/2}$: $| B_q(c, r) \cap C | \leq n^2$.
  Application: si taux erreur $p < 1- (1-d)^{1/2}$: le nbre de mots de code à distance $\leq p.n$ est $n^{O(1)}$.
- Donc, si p tend vers 1, on peut faire un décodage par liste (mais pas un décodage unique).

Existence de codes (p,L) décodables en liste

- Soit taux d'erreur et $\mathbf{H(p) = H_q(p)} = p.\log_q(q-1) - p.\log_q p - (1-p).\log_q(1-p)$ entropie q-aire
- Remarque: $n.H_q(p)$ est proche de $\log_q |B_q(0, p.n\ )|$:
  $$\log_q(\ n.H_q(p) - o(n)\ ) \leq \log_q(|B_q(0, p.n\ )|) \leq n.H_q(p).$$

- **Théorème**: Pour q, $L \geq 2$ et $\forall p \in]0,1-q^{-1}[$, $\exists$ famille de codes q-aires qui sont (p, L) décodables en liste et de rendement $R \geq 1- H_q(p) - 1/L$.
  - *Preuve*: basée sur un codage aléatoire de longueur n grande. On construit un code aléatoire de longueur n à partir de M mots tirés aléatoirement (non nécessairement distincts): on fixe M pour que le code soit (p,L)-décodable avec une bonne probabilité.
  - Prob{L+1 mots fixés $\in B_q(0, p.n\ )$ } = $(|B_q(0, p.n\ )| / q^n)^{L+1} \leq q^{-n.(L+1).(1-H(p))}$.
    Donc Prob(échec)=Prob{L+1 mots de code $\in$ même boule de rayon p.n} $\leq C(M, L+1).q^n.q^{-n.(L+1).(1-H(p))}$.
    Soit $\rho = 1 - H_q(p) - (L+1)^{-1}$; en posant $M=q^{\rho n}$ on a : Prob(échec)< 1/(L+1)! < 1/3.
    Parmi les M mots, il y a au moins M/2 mots distincts avec prob > ½ (coupon collector).
    Le code formé par ces $M/2 \geq q^{\rho n}/2 \geq q^{n.(1-H(p)-1/L)}$ mots est (p-L)-decodable avec probabilité ≥2/3.
    Donc il existe nécessairement un code (p,L)-décodable avec M/2, donc un rendement $R \geq 1- H_q(p) - 1/L$.

- **Intérêt**: preuve non constructive, mais le rendement optimal pour le décodage en liste jusqu'à un rayon p est $1-H_q(p)$, la capacité du canal q-aire.

Existence de codes *linéaires* (p,L) décodables en liste

- **Théorème**: Pour q puissance d'un entier premier, $\forall p \in]0,1-q^{-1}[$ et $\forall L \geq 2$, $\exists$ famille de codes q-aires **linéaires** qui sont (p, L) décodables en liste et de rendement $R \geq 1- H_q(p) - 1/\log_q(L+1)$.
  - *Preuve*: non constructive, en tirant au hasard un code linéaire (ie des mots linéairement dépendants).

- **Implication pour de grands alphabets :**
  On a $H_q(p) = p.\log_q(q-1) - p.\log_q p - (1-p).\log_q(1-p)$
  $= p - p.\log_q(q/(q-1)) + H_2(p)/\log_2 q.$

  Donc, pour q grand, et pour L grand, il existe des codes linéaires décodables en liste de rendement proche de $1 - H_q(p) \simeq 1 - p$.
  - Autrement dit: décodage en liste avec rendement R jusqu'à une fraction d'erreur (1-R)
  - De plus: la capacité de canal est $1 - H_q(p) \geq 1 - p - 1//\log_2 q$: donc on peut, avec q suffisamment grand, obtenir un code de rendement 1-p-ε pour décoder en liste jusqu'à une fraction p d'erreurs.

Décodage de codes de Reed-Solomon

- Code RS(n , k+1, distance = n-k) sur F, supposé de cardinal $q \geq n$.
  - évaluation d'un polynôme de degré k aux n abscisses: $\alpha_1, \ldots, \alpha_n$ de F

- **Problème jouet 1 : Décodage de 2 mots mélangés avec $n \geq 2k+1$:**
  - Entrée: 2 mots $a_1, \ldots, a_n$ et $b_1, \ldots, b_n \in F^n$ vérifiant $\exists P_1, P_2 \in F[X]$ de degré k tels que
    $\forall i$: $\{ a_i = P_1(\alpha_i)$ et $b_i = P_2(\alpha_i) \}$ ou $\{ b_i = P_1(\alpha_i)$ et $a_i = P_2(\alpha_i) \}$.
  - Sortie : les coefficients de P_1 et P_2.
- Algorithme:
  - On a: $P_1(\alpha_i) + P_2(\alpha_i) = a_i + b_i$ et $P_1(\alpha_i).P_2(\alpha_i) = a_i.b_i$
  - Par interpolation, on peut donc calculer $S=P_1+P_2$ et $P=P_1.P_2$ (de degrés k et 2k).
    Soit alors $Q(X, Y) = (Y - P_1(X)).(Y - P_2(X)) = Y^2 - S(X).Y + P(X)$.
  - La factorisation de Q en facteurs irréductibles (Q de degré 2 en Y) donne P_1 et P_2.

Décodage de codes de Reed-Solomon

- **Problème jouet 2 : Décodage d'un mot mélangé avec n ≥ 6k+1 :**
  - Entrée: un mot $y_1, \ldots, y_n \in F^n$ vérifiant $\exists P_1, P_2 \in F[X]$ de degré k tels que
    $\forall i : \; y_i = P_1(\alpha_i)$ ou $y_i = P_2(\alpha_i)$
    et de plus pour j= 1, 2: $\#\{i / y_i = P_j(\alpha_i)\} \geq n/3$ [ie au moins 2k valeurs pour P_1 et P_2]
  - Sortie : les coefficients de P_1 et P_2.
    - Remarque:
      - Soit $Q(X, Y) = (Y - P_1(X)).(Y - P_2(X))$: on a alors $\forall i : \; Q(\alpha_i, y_i) = 0$.
      - Mais ces équations ne définissent pas Q de manière unique !
- **Algorithme**: On cherche Q de la forme $Q(X,Y) = Y^2 - (\Sigma_{j=0}^{k} q_{1,j}.X^j).Y + (\Sigma_{j=0}^{2k} q_{2,j}.X^j)$: les 6k+1 équations $Q(\alpha_i, y_i) = 0$ donnent un système linéaire en les 3k+2 inconnues $q_{\_,\_}$. Soit Q(X,Y) la solution (unique) de ce système.
  - *Lemme*: $Q(X,P_1(X)) = 0$ autrement dit $(Y - P_1(X))$ est un facteur de Q [de même P_2].
    - Preuve: $R(X) = Q(X,P_1(X))$: R est de degré 2k < n/3.
      Soit $I=\{i \text{ tq } y_i = P_1(\alpha_i)\}$: $\forall i \in I \; R(\alpha_i)=0$; donc R a au moins n/3 racines. D'où R=0.
  - Donc la solution Q vérifie $Q(X, Y) = (Y - P_1(X)).(Y - P_2(X))$: la factorisation de Q par rapport à Y donne alors P_1 et P_2.

Décodage en liste de codes de Reed-Solomon

- **Problème reconstruction polynomiale avec ≤e erreurs, t=n-e:**
  - Entrée: $y_1, \ldots, y_n \in F^n$ vérifiant $\exists P \in F[X]$ de degré k tel que $\#\{i / y_i = P(\alpha_i)\} \geq t$.
  - Sortie : la liste de tous les polynômes P tels que que $\#\{i / y_i = P(\alpha_i)\} \geq t$.
- Généralisation lemme précédent: soit $Q(X,Y) = \Sigma_{j,j} q_{i,j}.X^i.Y^j$.
  - Déf: Degré (1,k)-pondéré : $wdeg_{1,k}(Q(X, Y)) = \text{Max}\{ i+kj \text{ tq } q_{i,j} \neq 0\}$
  - **Lemme**: Soit $Q(X,Y)\neq 0$ tel que $wdeg_{1,k}(Q) < t$ et $\forall i : \; Q(\alpha_i, y_i) = 0$. Soit P(X) de degré k tel que $\#\{i / y_i = P(\alpha_i)\} \geq t$. Alors Q(X, P(X))=0, ie (Y-P(X)) facteur de Q(X,Y).
    - *Preuve*: R(X)=Q(X, P(X)) est de degré $\text{Max}\{i+kj \text{ tq } q_{i,j} \neq 0\} < t$ et s'annulle en ≥ t valeurs α_i; d'où R=0.
  - Donc: il suffit de montrer l'existence de Q, puis de résoudre un système linéaire.
- **Théorème**: Si $(D+2)(D+1)>2.k.n$ alors $\exists Q(X,Y)\neq 0$ tel que $wdeg_{1,k}(Q)\leq D$ et $\forall i \; Q(\alpha_i, y_i)=0$.
  - *Preuve*: Comme $wdeg_{1,k}(Q)\leq D$, on a $Q(X,Y) = = \Sigma_{j=0}^{D/k} \Sigma_{i=0}^{D-jk} q_{i,j}.X^i.Y^j$. (car degré ≤ D/k en y et ≤D en X).
    Les n équations $Q(\alpha_i, y_i)=0$ donnent un système linéaire à n équations et m inconnues, les $q_{i,j}$.
    Si m>n, alors on a (au moins) une solution non nulle. Or $m = \Sigma_{j=0}^{D/k} \Sigma_{i=0}^{D-jk} 1 = \Sigma_{j=0}^{D/k} (D - jk + 1) \geq (D+1)(D+2) / (2.k)$ qed.
- **Algorithme**: Soit t tel que $t^2 > 2.k.n$ le nombre de valeurs correctes et soit $D=[(2.k.n)^{1/2}]$.
  - Étape 1: résoudre le système linéaire pour calculer $Q(X,Y)\neq 0$ 0 tel que $wdeg_{1,k}(Q)\leq D$ et $\forall i \; Q(\alpha_i, y_i)=0$.
  - Étape 2: (factorisation / calcul de racines en Y de Q(X,Y) pour trouver tous les polynômes P(X) de degré k tels que Q(X, P(X))=0; pour chacun vérifier que $\#\{i / y_i = P(\alpha_i)\} \geq t$.
  - Coût: système lin=$O((n+D)^3)$. De plus, le nombre de facteurs P(X) possibles de Q(X,Y) est ≤ D/k ≤ $\sqrt{(2n/k)}$.

Conclusion Décodage en liste de codes de Reed-Solomon

- Si le nombre de valeurs correctes t vérifie $t > \sqrt{(2.n.k)}$, l'algorithme est de coût polynomial et retourne une liste de candidats de taille $\leq \sqrt{(2.n/k)}$.
  - Donc, avec $k = \Omega(n)$, une liste de taille $O(1)$

- Le décodage en liste d'un code Reed-Solomon(n, k+1, n-k) de rendement $R = (k+1)/n$:
  - Peut être fait et garanti jusqu'à une fraction d'erreurs $= 1 - \sqrt{(2.R)}$ et retourne moins de $\sqrt{(2/R)}$ candidats.

  - En particulier: avec un rendement faible, on peut corriger jusqu'à presque 100% d'erreurs.
    - Utile en cryptography : exemple pour construire des "prédicats à sens unique" àpartir de fonctions à sens unique, utilisés dans les générateurs pseudo-aléatoires cryptographiquement non-prédictibles (CSPRNG).

- Extension au décodage unique: si #erreurs $\leq (d-1)/2 < (n-k)/2$, alors on a $t > (n+k)/2$; on interpole $Q(X,Y) \neq 0$ en le cherchant sous la forme :
  $Q(X,Y) = A(X).Y + B(X)$ avec degré$(A) < (n-k-1)/2$ et degré$(B) < (n+k-1)/2$.
  Le polynôme corrigé est alors $-B(X)/A(X)$.
  - *Preuve*: Soit f (X) la solution unique et e(X) le polynôme localisateur d'erreur.
    Le choix $A(X)=e(X)$ et $B(X)=-f(X)/e(X)$ donne un polynôme Q(X,Y) qui convient et $wdeg_{1,k}(Q) \leq (n-k-1)/2 + k < t$.

V Exemples de codes utilisés dans des matériels

CDROM / DVDROM (1/3)

- Codes par bloc cycliques
- Exemple : CD Audio :
  - Données = octets
  - K=trame = 24 octets codés sur 32 octets sur le CD
  - CIRC = Code de Reed-Solomon entrelacé croisé
    - Code (32, 24) = entrelacement de 2 codes cycliques
  - Base:
    - code de Reed-Solomon (255, 251, 5)
    - Code C1(28,24, 5)
    - Code C2(32,28,5)
  - Codes en détail
  - Description de l'entrelacement

CDROM : code CIRC (2/3)

Le code C_1

Il s'agit d'un code (28,24,5) sur le corps à 256 éléments.

Si x est un mot de 24 octets le mot de code de C_1 qui lui correspond est le mot de 28 octets égal à $(x, x\,R_1{}^t)$, où la matrice de dimension (4,24) R_1 est définie par

$$R_1 = \begin{bmatrix} a^6 & a^{192} & a^{142} & a^{159} & a^{99} & a^{88} & a^{104} & a^{144} & a^{55} & a^{180} & a^{174} & a^{101} & a^{111} & a^{118} & a^{169} & a^{107} & a^{132} & a^{25} & a^{167} & a^{239} & a^{168} & a^{188} & a^{11} \\ a^{45} & a^{108} & a^{248} & a^{131} & a^{64} & a^{221} & a^{100} & a^{235} & a^{147} & a^{45} & a^{198} & a^{21} & a^{228} & a^{186} & a^{231} & a^{56} & a^{68} & a^{81} & a^{46} & a^{32} & a^{60} & a^{225} & a^{13} \\ a^{50} & a^{52} & a^{59} & a^{132} & a^{186} & a^{81} & a^{128} & a^{126} & a^{133} & a^{32} & a^{213} & a^{195} & a^{43} & a^{198} & a^{194} & a^{13} & a^{167} & a^{167} & a^{252} & a^{61} & a^{3} & a^{12} & a^{6} \\ a^{42} & a^{136} & a^{153} & a^{93} & a^{82} & a^{98} & a^{138} & a^{49} & a^{174} & a^{168} & a^{95} & a^{105} & a^{112} & a^{163} & a^{101} & a^{126} & a^{19} & a^{161} & a^{233} & a^{162} & a^{182} & a^{105} & a^{3} \end{bmatrix}$$

Le code C_2

Il s'agit d'un code (32,28,5) sur le corps à 256 éléments.

Si x est un mot de 28 octets le mot de code de C_2 qui lui correspond est le mot de 32 octets égal à $(x, x\,R_2{}^{tr})$, où la matrice $(4,28) R_2$ est :

$$R_2 = (R_1 \mid R') \text{ où } R' = \begin{bmatrix} a^{232} & a^{98} & a^{54} & a^{174} \\ a^{167} & a^{211} & a^{180} & a^{143} \\ a^{24} & a^{41} & a^{188} & a^{164} \\ a^{92} & a^{48} & a^{168} & a^{67} \end{bmatrix}$$

CDROM : code CIRC (3/3)

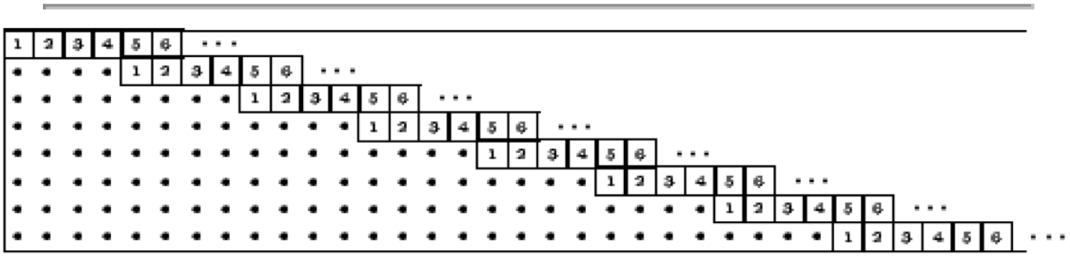


Figure 1: Table d'entrelacement à retard 4 de profondeur 8

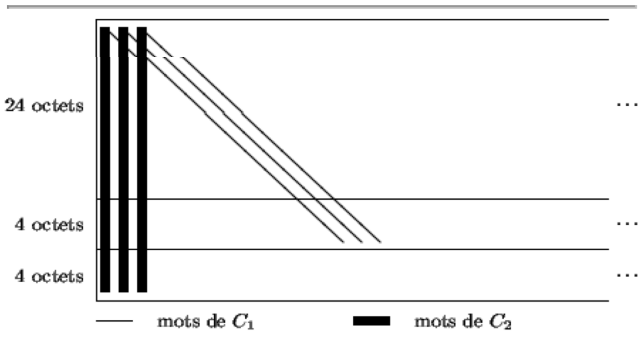


Figure 2: Schéma de codage

Satellites : ex. Voyager

- Photos de Saturne et Jupiter (1977)
  - Données peu critiques (images 800x800 - 8 bits)
  - Données critiques: GSE (General Science&Engineering)
    - Mesures
    - Contrôle
- Données GSE codées par Golay(24,12) sur F_2 (6-correcteur)
- Autres données: code convolutif

GSM

- Signal parole: par tranche de 20 ms
- Codec: Numérisation : 260 bits
  = 50 très critiques + 132 critiques + 78 complémentaires
- Codage de canal : 456 bits
  - » 50 bits : CRC (X3+x+1)
  - » 182+3 : codage convolutif : x2 : 378 bits
  - » +78= 4 bits controles :
- Entrelacement (diagonal : sur plusieurs trames de 456 bits)
- Chiffrement / modulation

Autres codes et applications

Codes Goppa(n,k,d) : définis par une courbe f(x,y) sur un GF

- n=nombre de points de f et d ≥ n-deg(f); ex: $x^3y+y^3+x=0$ ➔ Goppa(24,3,20)
- Pour l'instant difficiles à rendre efficaces mais très grande distance minimale

Turbo codes [C. Berrou, A. Glavieux] : codes de convolution

- fonctionnent seulement sur un petit alphabet (n et k ≤ 8)
- Décodage est complexe mais la correction est très grande

Cryptographie : système McEliece : code linéaire 2^{500} mots

- Problème à sens unique : trouver un mot de poids minimal
  - Facile si la matrice génératrice normalisée est connue
  - Difficile (énumérer tous les mots !) si la matrice est mélangée
  - Clef publique : matrice génératrice permutée
  - Clef privée : matrice génératrice normalisée