Librairie Perseus : les enjeux d'une alternative au chiffrement

Eric Filiol
filiol@esiea.fr

ESIEA - Laval
Laboratoire de cryptologie et de virologie opérationnelles $(C+V)^O$

RMLL 2011

Introduction

- La protection des informations et des données privées/confidentielles devient une préoccupation majeure :
  - Emergence des officines de renseignement privées.
  - Evolution vers le contrôle des citoyens à des fins commerciales (ex. : HADOPI), politiques (traque des dissidents), économiques (espionnage industriel) ....
- La quasi-totalité de nos flux d'information est désormais sous surveillance.
- La mobilité accroît le risque (smartphone, laptop...).
- Situation très critique dans certains pays dictatoriaux (Chine, Myanmar, Iran...) voire démocratique (USA, UK...).

Introduction (2)

- Necessité impérieuse de préserver la sécurité des Etats et des citoyens (missions régaliennes).
- La question est : comment empêcher les abus en termes d'interception des données personnelles/privées tout en
  - préservant les capacités techniques des Etats dans le cadre de leurs nécessaires missions régaliennes (sécurité intérieure, défense du territoire, lutte contre la criminalité, la pédopornographie...) ,
  - respectant les diverses lois nationales en matière d'utilisation et de diffusion de moyens de protection des données,
  - ET tout en préservant le droit naturel des citoyens à communiquer librement et de manière privée.
- Une solution est la technologie PERSEUS.

Introduction (3)

Solution classique : la cryptographie. Mais problèmes légaux et graves préjudices envers la sécurité (interne, externe) des Etats. De plus la cryptographie se détecte immédiatement (absence de TRANSEC).

- Pour gérer toutes ces contraintes (techniques, légales...) il nous faut
  - un procédé qui ne peut être cassé QUE grâce à une puissance de calcul hors norme et pendant un temps significativement long (typiquement un supercalculateur tournant pendant plusieurs jours/semaines/mois) ;
  - sinon il est incassable en pratique,
  - le procédé doit être lui-même difficile à détecter (TRANSEC).
- Cette solution limite naturellement les tentatives d'écoutes abusives.
- La solution : remplacer la cryptographie par des techniques de codage et du bruit maitrisé.

Introduction (4)

- Exemple classique : journaliste occidental en Chine.
  - Il écrit ses articles (sur les Droits de l'Homme en Chine) et les envoie de Chine.
  - Les chinois peuvent éventuellement casser le message (s'ils ont pu l'identifier et le repérer) mais seulement après que le journaliste soit reparti dans son pays.
- Autres cas : dissidents, industriels à l'étranger ...

Chiffrement contre Codage bruité

- La définition "légale" de la cryptographie est en faite directement liée à la probabilité suivante :

$$P[c_t = m_t + e_t] = P[e_t = 1]$$

où c_t, m_t décrivent respectivement les bits de texte chiffré et de clair et où e_t peut être décrit par le bruit introduit, par la clef, lors du chiffrement, et le procédé de chiffrement lui-même, à l'instant t.

  - Si $P[e_t = 1] = \frac{1}{2} + \epsilon$ (avec ϵ très proche de 0) alors il s'agit de cryptographie.
  - Sinon (ϵ significativement différent de 0), il s'agit de codage bruité.

- L'approche consiste alors à considérer un problème calculatoirement difficile (pour l'attaquant) issu de la théorie des codes.

Chiffrement contre Codage bruité (2)

- Avantage des données bruitées.
  - Les données chiffrées exhibent un profil entropique maximal donc très facilement identifiable.
  - Les données codées bruitées ont en revanche un profil entropique bas, proche de celui de communication normale (par exemple communications téléphoniques mobiles).
- Ce profil entropique bas permet de contourner toutes les techniques de détection reposant sur l'entropie ou les approches statistiques,
  - Mise en œuvre de propriétés TRANSEC (noyer des données codées parmi les nombreuses autres données codées).
  - Contournement de filtres de type Echelon, firewalls, IDS....

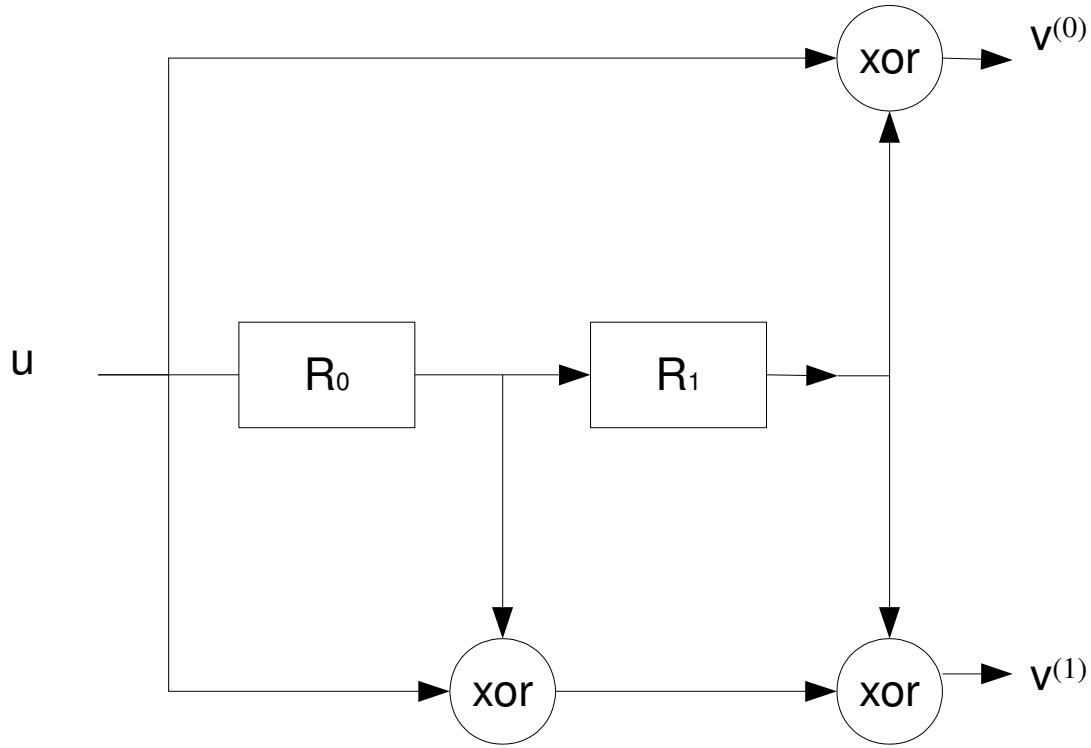
Plan

Codes correcteurs d'erreurs

- PERSEUS utilise une variété de codes correcteurs d'erreurs très répandus : les codes convolutifs poinçonnés.
- Omniprésents dans la téléphonie mobile (GSM, UMTS, GPRS...), les communications satellites (turbo-codes)...
- Taux de bruit admissibles : moins de 1 %, le décodage étant calculatoirement complexe.
- Sauf exception (ex. : armée Tchèque) les codeurs sont connus et les paramètres ont des valeurs faibles.

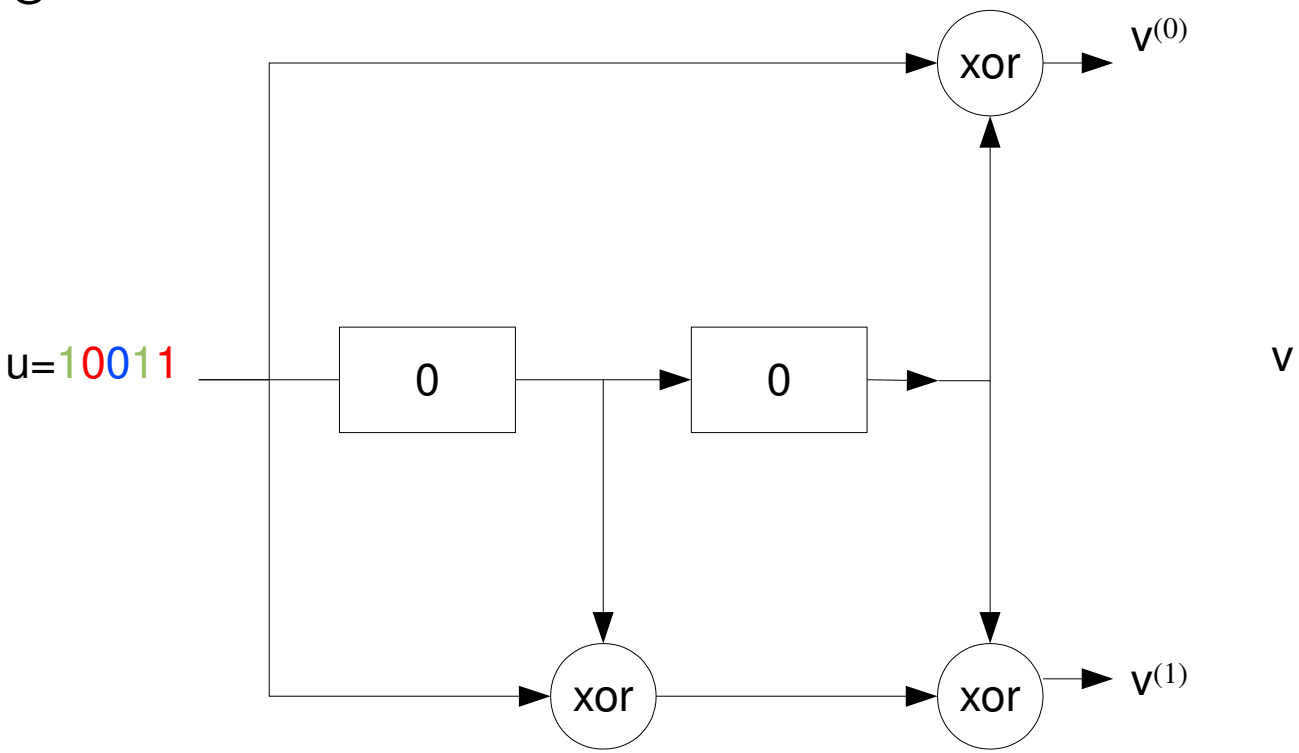
Code convolutif

Soit un code convolutif $\mathcal{C}$ de taux $\frac{1}{2}$ et d'une taille mémoire de $M = 2$.



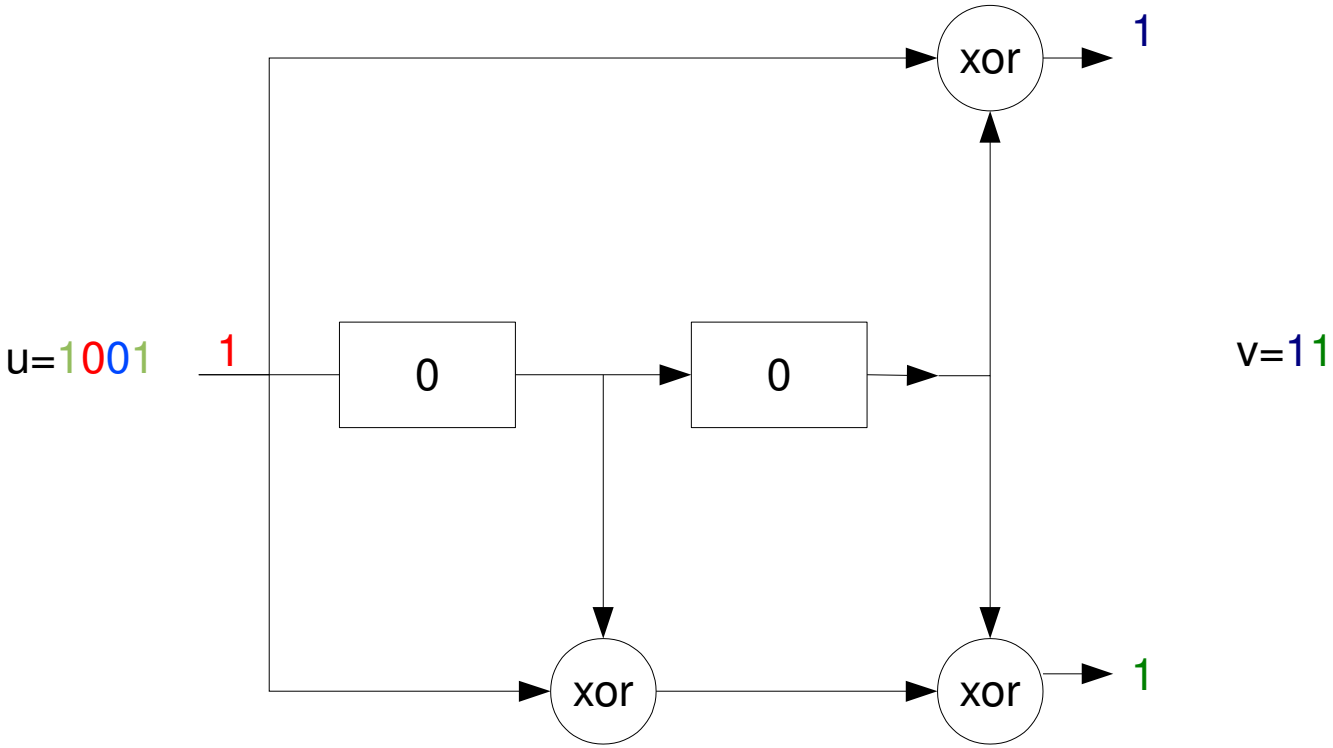
Code convolutif

Soit un code convolutif $\mathcal{C}$ de taux $\frac{1}{2}$ et d'une taille mémoire de $M = 2$.
Un message $u = 10011$



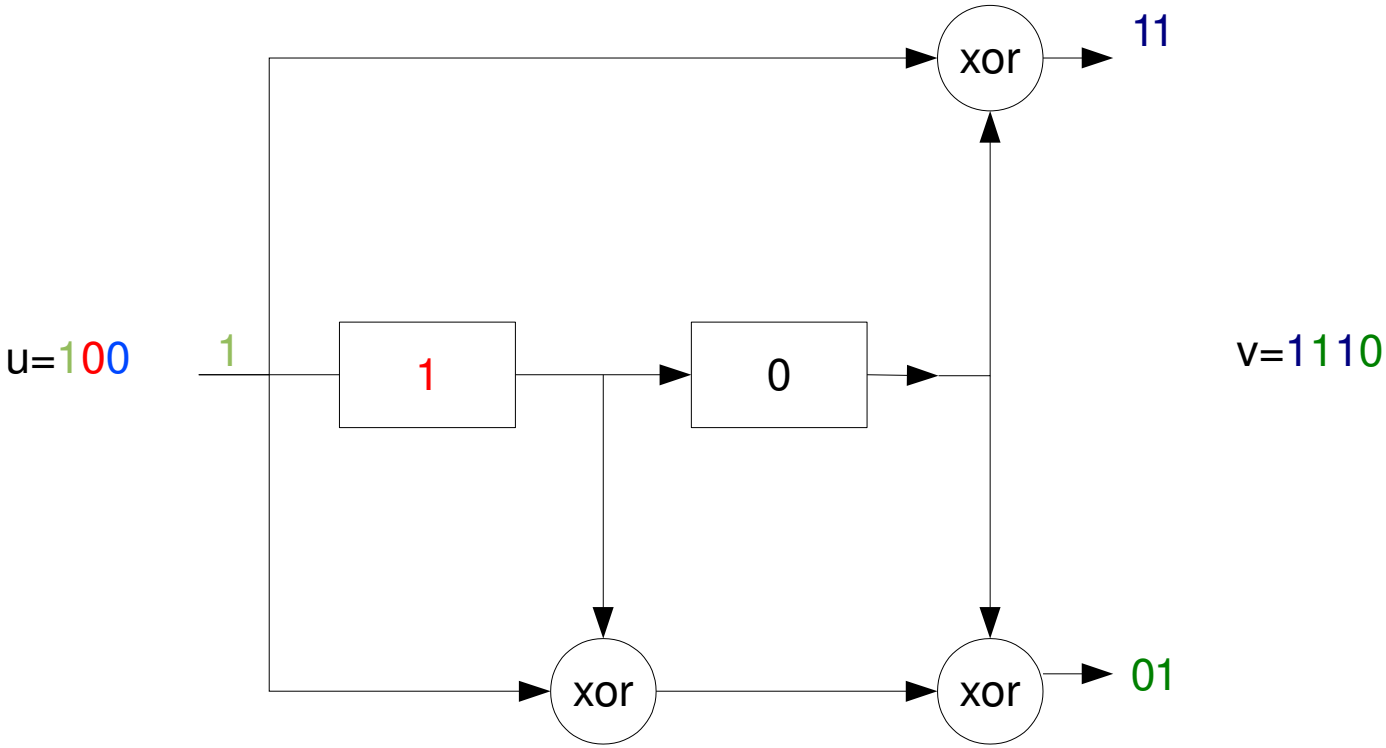
Code convolutif

Soit un code convolutif $\mathcal{C}$ de taux $\frac{1}{2}$ et d'une taille mémoire de $M = 2$.
Un message $u = 10011$



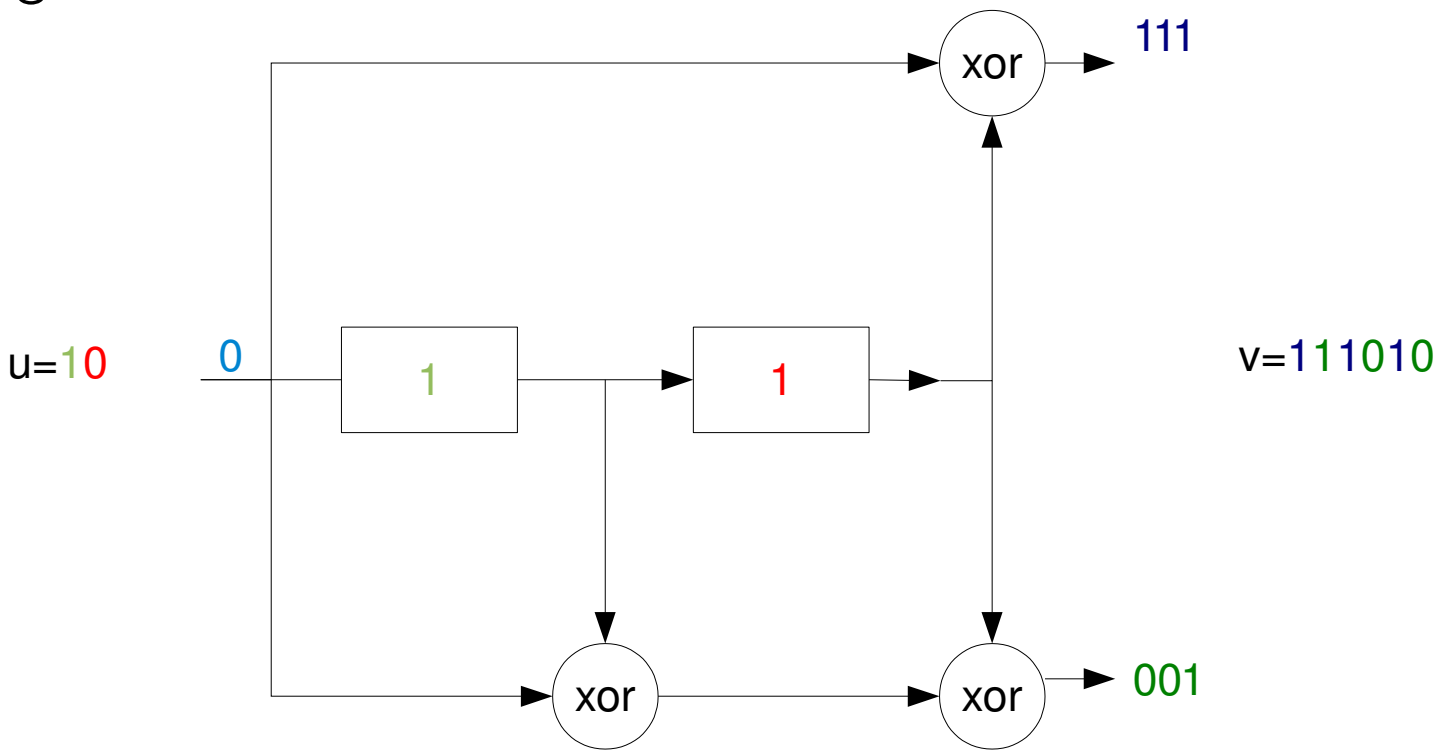
Code convolutif

Soit un code convolutif $\mathcal{C}$ de taux $\frac{1}{2}$ et d'une taille mémoire de $M = 2$.
Un message $u = 10011$



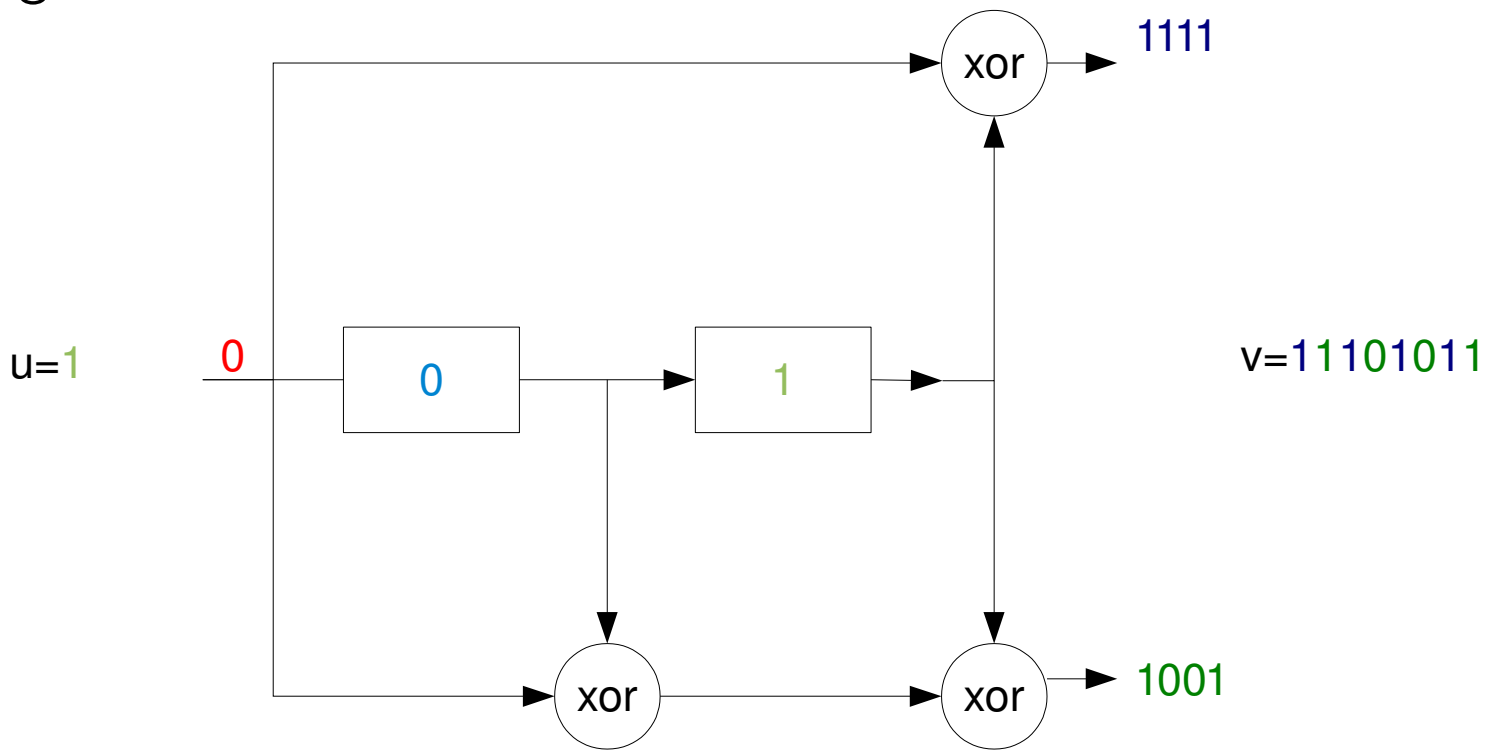
Code convolutif

Soit un code convolutif $\mathcal{C}$ de taux $\frac{1}{2}$ et d'une taille mémoire de $M = 2$.
Un message $u = 10011$



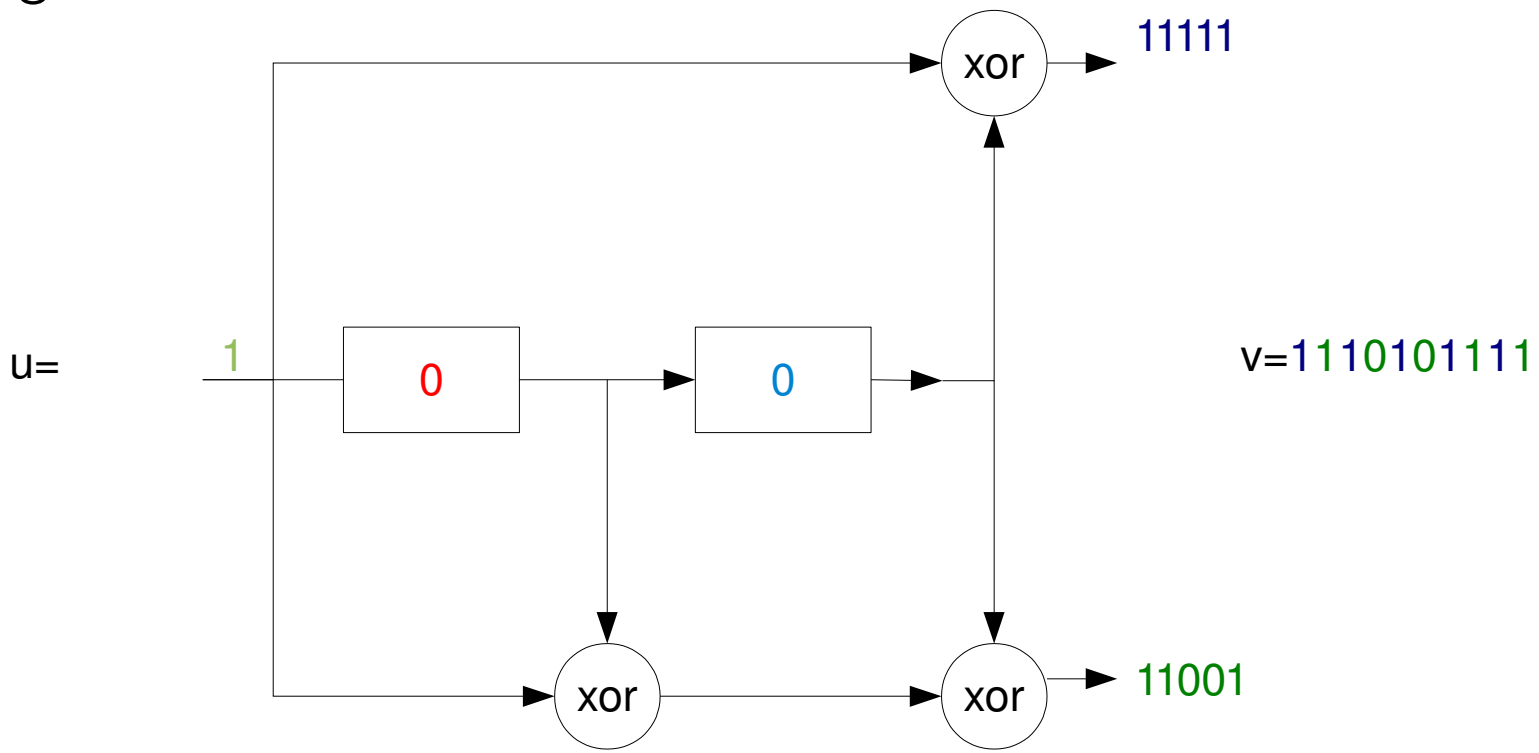
Code convolutif

Soit un code convolutif $\mathcal{C}$ de taux $\frac{1}{2}$ et d'une taille mémoire de $M = 2$.
Un message $u = 10011$



Code convolutif

Soit un code convolutif $\mathcal{C}$ de taux $\frac{1}{2}$ et d'une taille mémoire de $M = 2$.
Un message $u = 10011$



Présentation

Un code convolutif est défini par

- un taux : $\frac{k}{n}$
- une taille de mémoire (ou longueur de contrainte) $K = M + 1$.

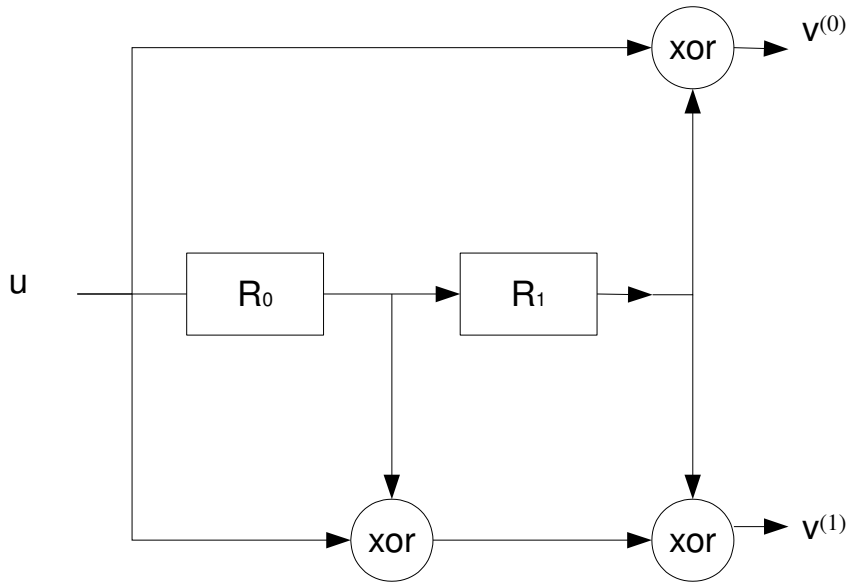
Notation

(n, k, K) code convolutif

Vision alternative

Code Convolutif

k registres avec n polynomials agissant sur chaque registre.
au total $n \times k$ polynômes pour un (n, k, K) code convolutif.
Le degré de chaque polynôme est égal à $K - 1$.



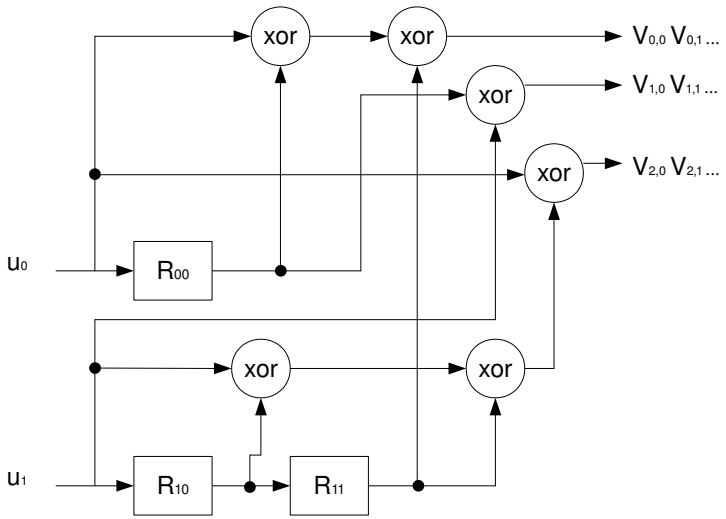
$\mathcal{C}$: $(2, 1, 3)$ code

$v_0 : 1 + x^2$
$v_1 : 1 + x + x^2$

Vision alternative

Code Convolutif

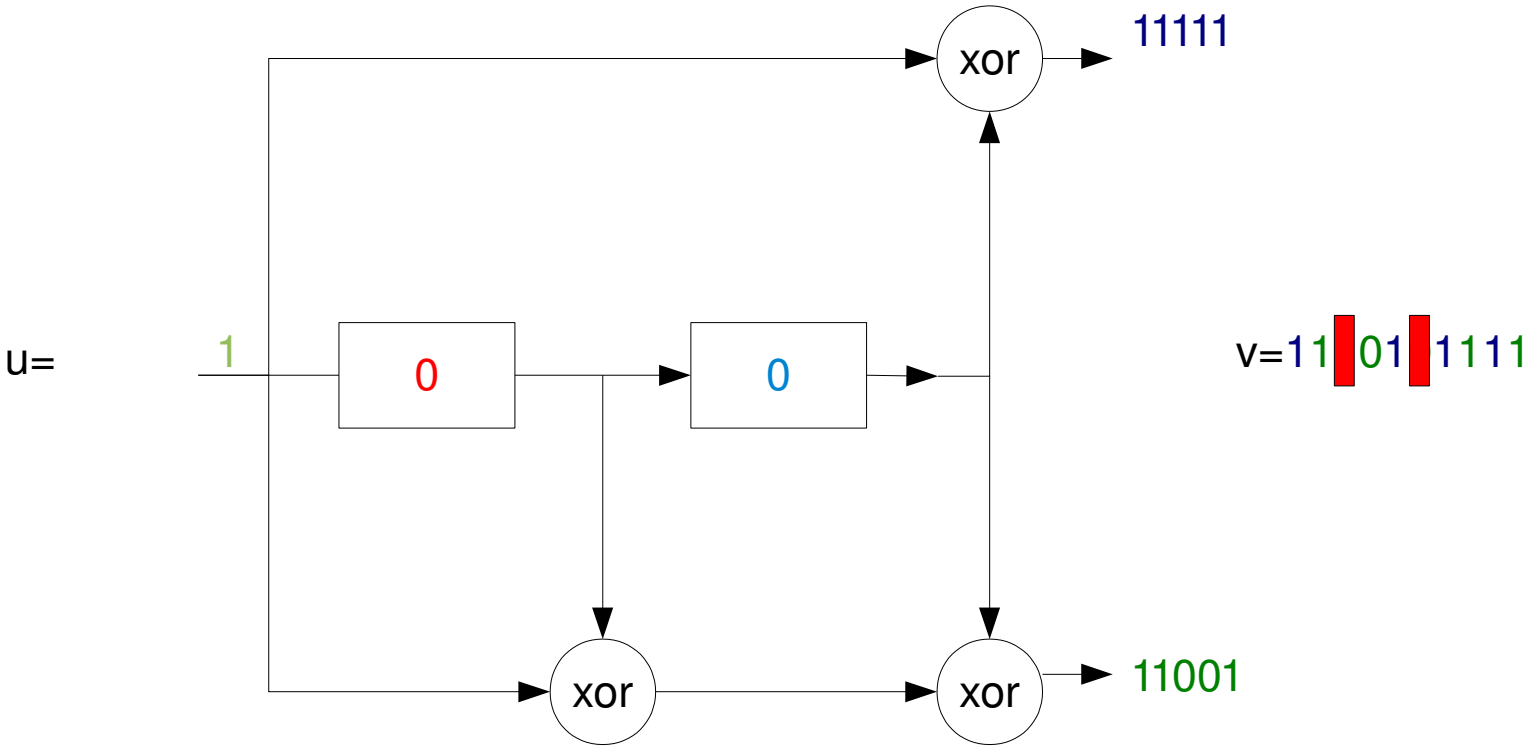
k registres avec n polynomials agissant sur chaque registre.
au total $n \times k$ polynômes pour un (n, k, K) code convolutif.
Le degré de chaque polynôme est égal à $K - 1$.



$\mathcal{C}$: $(3, 2, 3)$ code

$v_{0,0} : 1 + x$
$v_{0,1} : x^2$
$v_{1,0} : x$
$v_{1,1} : 1$
$v_{2,0} : 1$
$v_{2,1} : 1 + x + x^2$

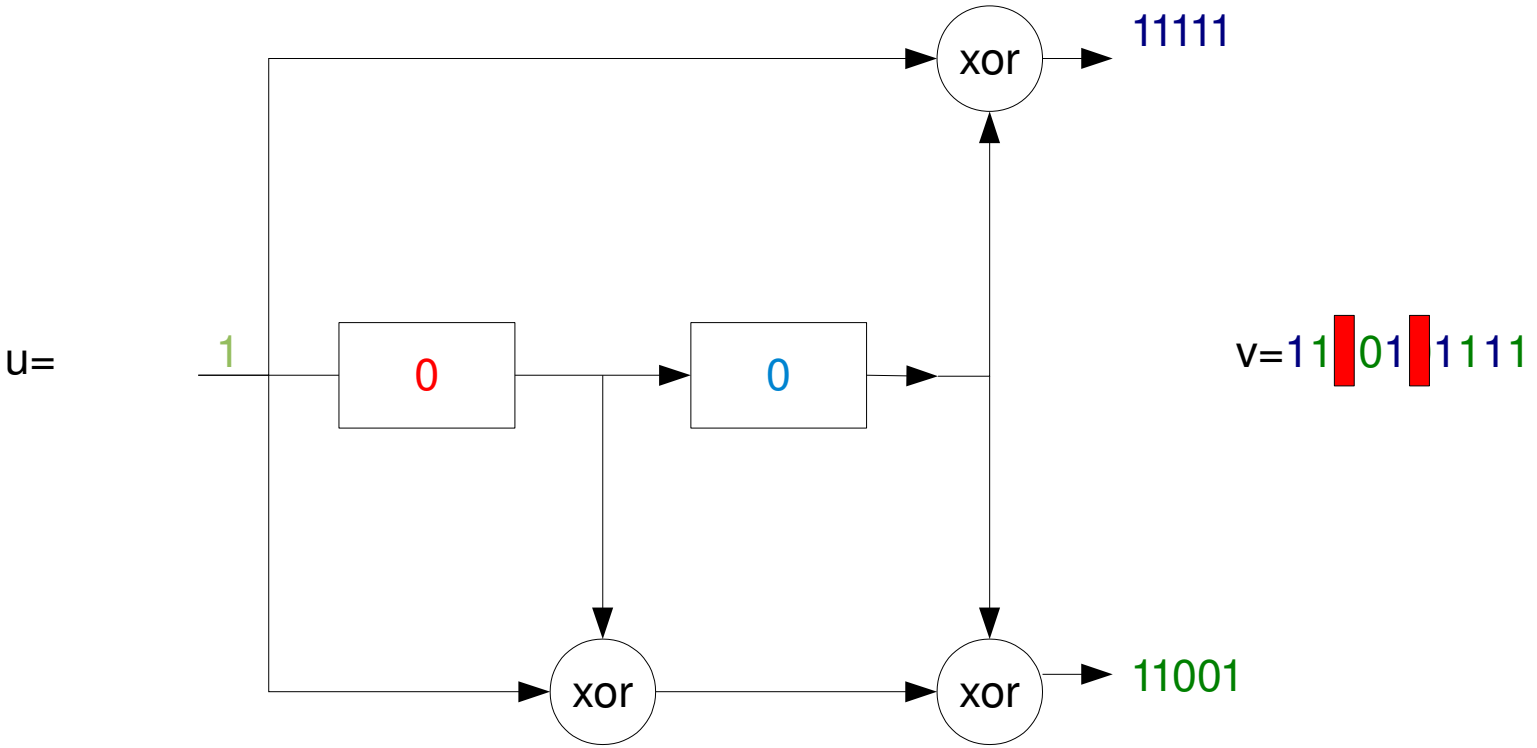
Poinçonnage



Motif de poinçonnage

P : une matrice de taille $J \times n$ et de poids I.

Poinçonnage



Motif de poinçonnage

P : une matrice de taille $J \times n$ et de poids I.

Exemple

Soit P un motif de poinçonnage défini par :

$$P = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$$

et soit v la séquence produite par un $(2, 1, 3)$ code :

$$v = \left(\begin{array}{cc|cc|c} 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 \end{array} \right)$$

$$\left(\begin{array}{cc|cc|c} 1 & & 1 & & 1 \\ 1 & 0 & 0 & 1 & 1 \end{array} \right) \Rightarrow 11010111$$

Pourquoi poinçonner ?

1. Economiser de la bande passante.
2. Produire un codeur équivalent (non poinçonné) plus complexe et plus difficile à reconstruire (voir plus loin).

Pourquoi poinçonner ?

1. Economiser de la bande passante.
2. Produire un codeur équivalent (non poinçonné) plus complexe et plus difficile à reconstruire (voir plus loin).

Code convolutif (non poinçonné) équivalent

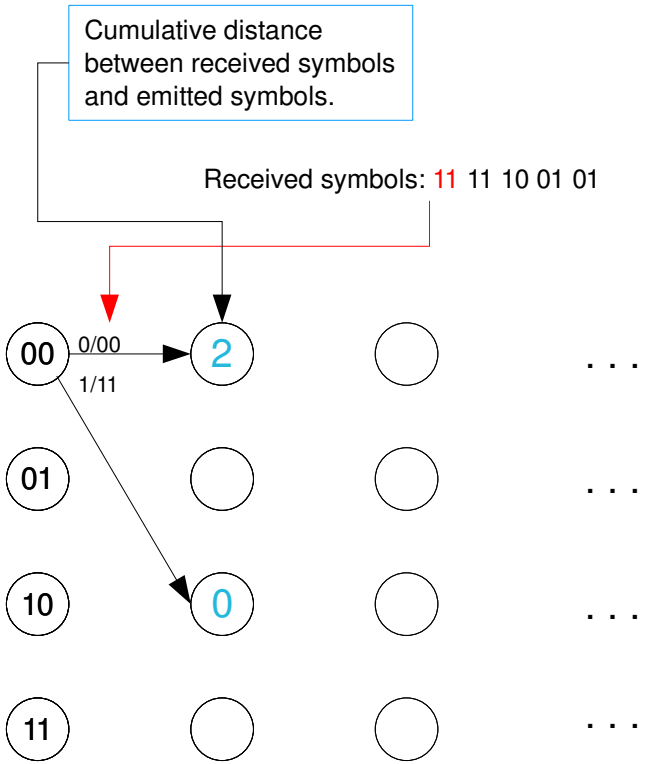
Un (n, k, K) code et une $J \times n$ matrice de poinçonnage P de poids I.

$$\Rightarrow (I, kJ, K) \text{ code convolutif non poinçonné}$$

Viterbi Algorithm

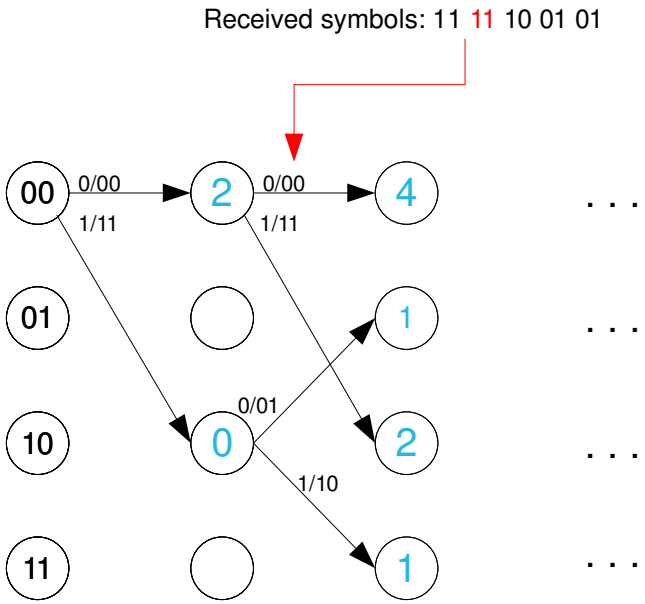
Viterbi Algorithm

Lattice construction



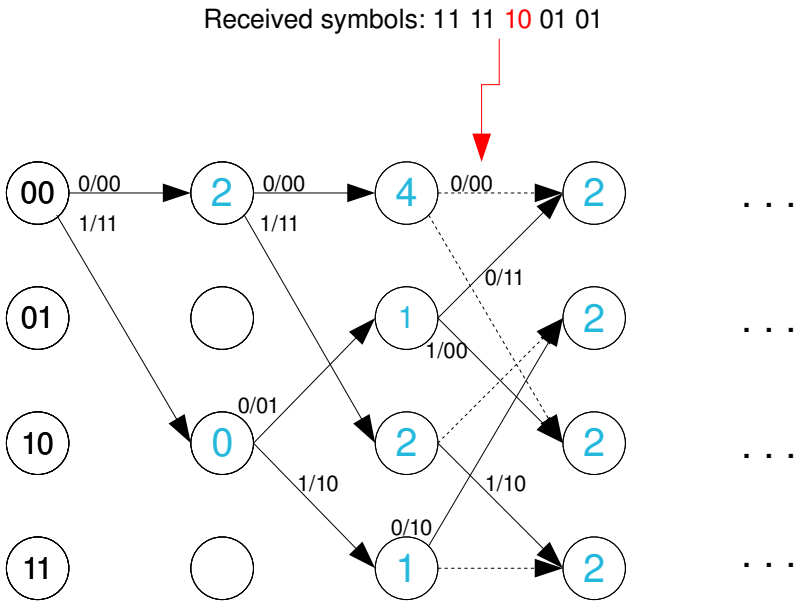
Viterbi Algorithm

Lattice construction



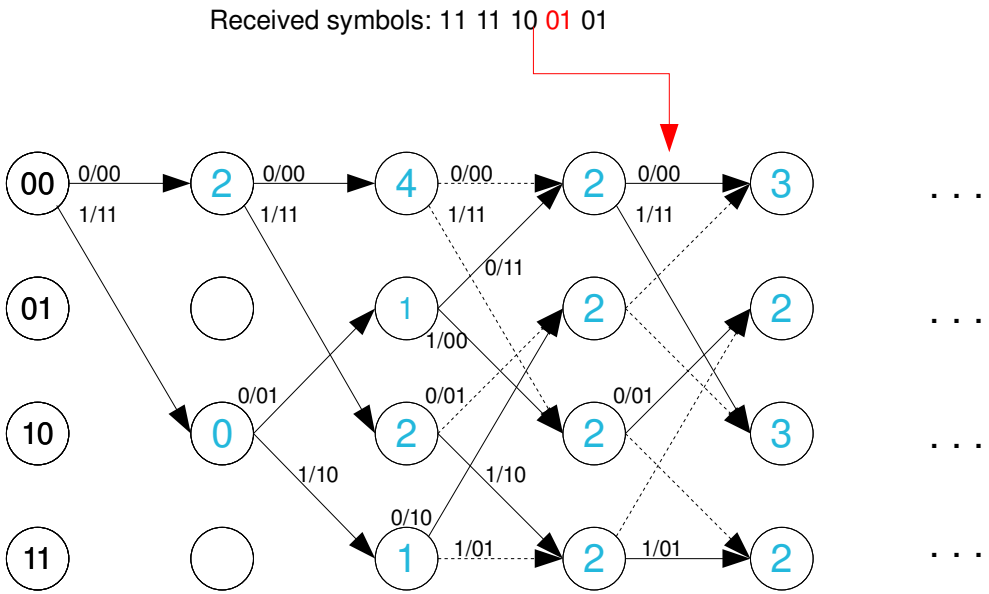
Viterbi Algorithm

Lattice construction



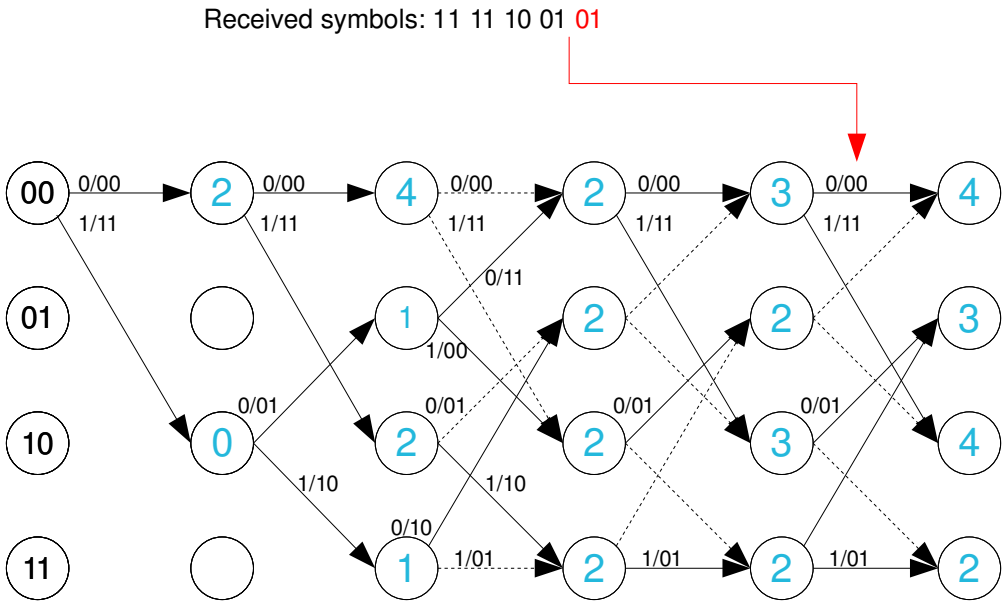
Viterbi Algorithm

Lattice construction



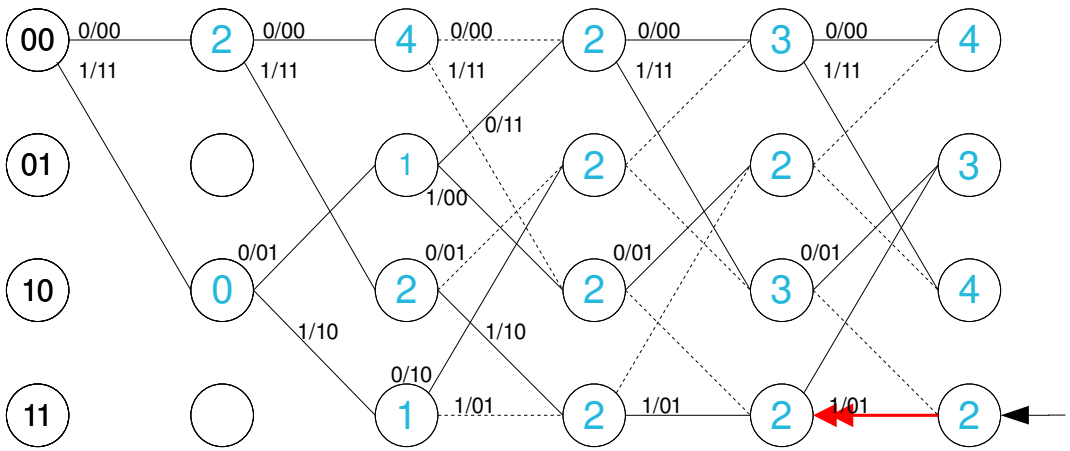
Viterbi Algorithm

Lattice construction



Viterbi Algorithm

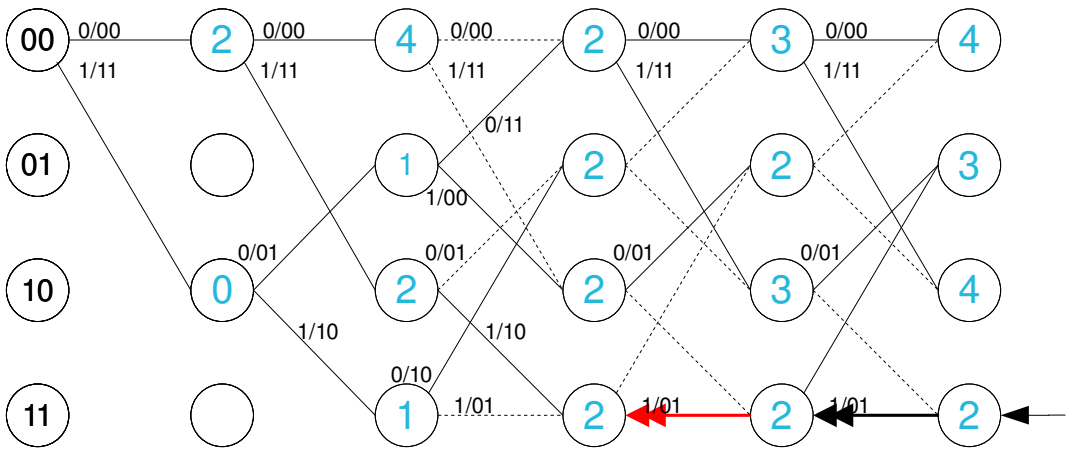
Backtracking



Decoded symbols: 1

Viterbi Algorithm

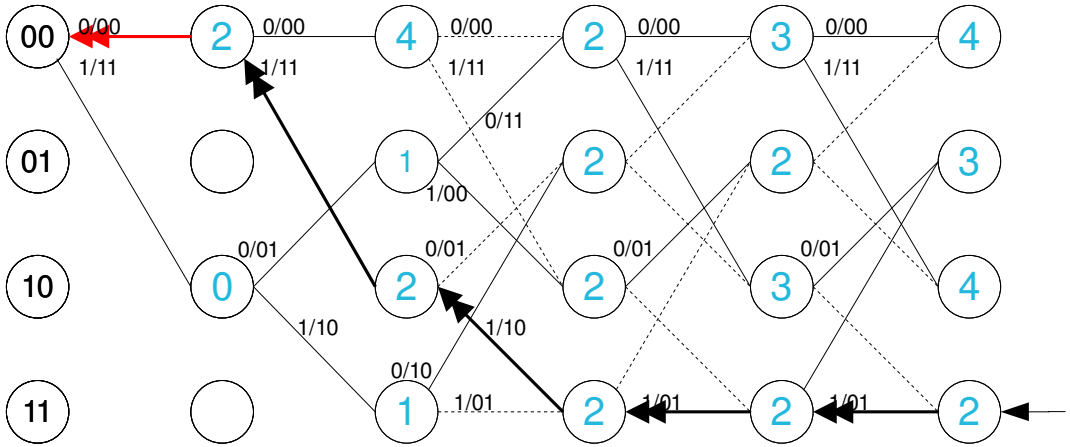
Backtracking



Decoded symbols: 11

Viterbi Algorithm

Backtracking



Decoded symbols: 01111

- Problème : le décodage a une complexité exponentielle en K.
- Gestion du poinçonnage : les bits absents sont remplacés par des zéros.

Reconstruction des codes convolutifs (Filiol 1997 - Barbier 2007)

But : retrouver tous les paramètres d'un codeur inconnu à partir des seules données codées, pour pouvoir ensuite décoder.

Puncturing effect

Soit un (n, k, K) code et une $J \times n$ matrice de poinçonnage P de poids I :

$$\Rightarrow (I, kJ, K) \text{ code convolutif}$$

La reconstruction a pour complexité

$$\mathcal{O}(\alpha \times n^5 \times K^4) \Rightarrow \mathcal{O}(\alpha \times I^5 \times K^4)$$

α : croit exponentiellement avec p, la probabilité de bruit.

Reconstruction des codes convolutifs (Filiol 1997 - Barbier 2007)

But : retrouver tous les paramètres d'un codeur inconnu à partir des seules données codées, pour pouvoir ensuite décoder.

Puncturing effect

Soit un (n, k, K) code et une $J \times n$ matrice de poinçonnage P de poids I :

$$\Rightarrow (I, kJ, K) \text{ code convolutif}$$

La reconstruction a pour complexité

$$\mathcal{O}(\alpha \times n^5 \times K^4) \Rightarrow \mathcal{O}(\alpha \times I^5 \times K^4)$$

α : croit exponentiellement avec p, la probabilité de bruit.

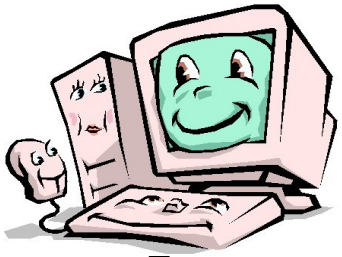
L'impact du bruit

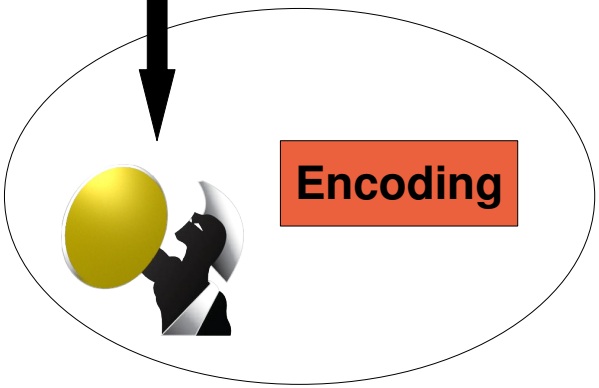
La probabilité de reconstruire le code avec succès décroit exponentiellement avec p. Dès que $p > 5\% \Rightarrow$ la reconstruction online est impossible calculatoirement.

Code	Temps de reconstruction ($p = 10^{-2}$)	Temps de reconstruction ($p = 2.10^{-2}$)
(4, 3, 8)	7 min 12 sec	Echec
(4, 3, 9)	6 min 16 sec	Echec

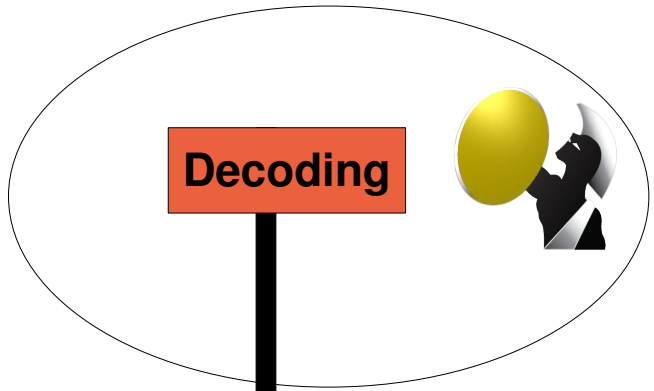
TABLE: Exemples de temps de calcul de reconstruction (Pentium IV 2.0 Ghz) pour deux niveaux de bruit

Plan





Encoding



Decoding

Principe général

- L'attaquant est confronté à un problème calculatoirement impossible à résoudre.
- Le codeur change aléatoirement et fréquemment (à chaque communication, chaque trame).
- Pour rendre la reconstruction calculatoirement complexe, ajout d'un bruit aléatoire déterministe généré à partir d'une quantité secrète commune (seuls émetteur et destinataire peuvent enlever le bruit).

Problem

Décodage de Viterbi facile tant que $p < 1 - 3\%$
Reconstruction impossible dès que $p > 5\%$ (inline mode)

Principe général

- L'attaquant est confronté à un problème calculatoirement impossible à résoudre.
- Le codeur change aléatoirement et fréquemment (à chaque communication, chaque trame).
- Pour rendre la reconstruction calculatoirement complexe, ajout d'un bruit aléatoire déterministe généré à partir d'une quantité secrète commune (seuls émetteur et destinataire peuvent enlever le bruit).

Problem

Décodage de Viterbi facile tant que $p < 1 - 3\%$
Reconstruction impossible dès que $p > 5\%$ (inline mode)

Solution

On ajoute un bruit déterministe aléatoire avec $p \in [10\%, 30\%]$

Pour chaque session ou trame

1. $5 < n \leq 12$
2. $1 < k < 6$
3. $10 < N \leq 30$
4. $n \times k$ polynômes de degré $N - 1$
5. Une $J \times n$-matrice P de poids $(n \times J) - (J - 1)$
6. X_0 une valeur de 128 bits (initialisation du générateur de bruit).

Pour chaque session ou trame

1. $5 < n \leq 12$
2. $1 < k < 6$
3. $10 < N \leq 30$
4. $n \times k$ polynômes de degré $N - 1$
5. Une $J \times n$-matrice P de poids $(n \times J) - (J - 1)$
6. X_0 une valeur de 128 bits (initialisation du générateur de bruit).

Pour chaque session ou trame

1. $5 < n \leq 12$
2. $1 < k < 6$
3. $10 < N \leq 30$
4. $n \times k$ polynômes de degré $N - 1$
5. Une $J \times n$-matrice P de poids $(n \times J) - (J - 1)$
6. X_0 une valeur de 128 bits (initialisation du générateur de bruit).

Gestion des paramètres

Protégés par un secret commun (clef) ou un protocole cryptographique (session HTTPS initiale)

Implémentation

- Ecrite en C (relativement) optimisé (pour rester lisible).
- Versions 1.x : décodeur Viterbi.
- Triple licence MPL/GPL/LGPL.
- Source, documentation disponible sur `http://code.google.com/p/libperseus`
- Bugs, retours, commentaires appréciés (`ffiliol@gmail.com`).

Structure de la librairie

Structure très simple : cinq procédures principales.

- int Gen_Pcc(PUNCT_CONC_CODE *) ;
- int Gen_Noise_Generator(NOISE_GEN *, INIT_NOISE_GEN *) ;
- int Gen_Noise(unsigned char *, NOISE_GEN *, unsigned long int, INIT_NOISE_GEN *) ;
- int PCC_Encode(unsigned char *, unsigned char *, PUNCT_CONC_CODE *, unsigned long int) ;
- int Viterbi_Decode(unsigned char *, unsigned char *, PUNCT_CONC_CODE *, unsigned long int) ;

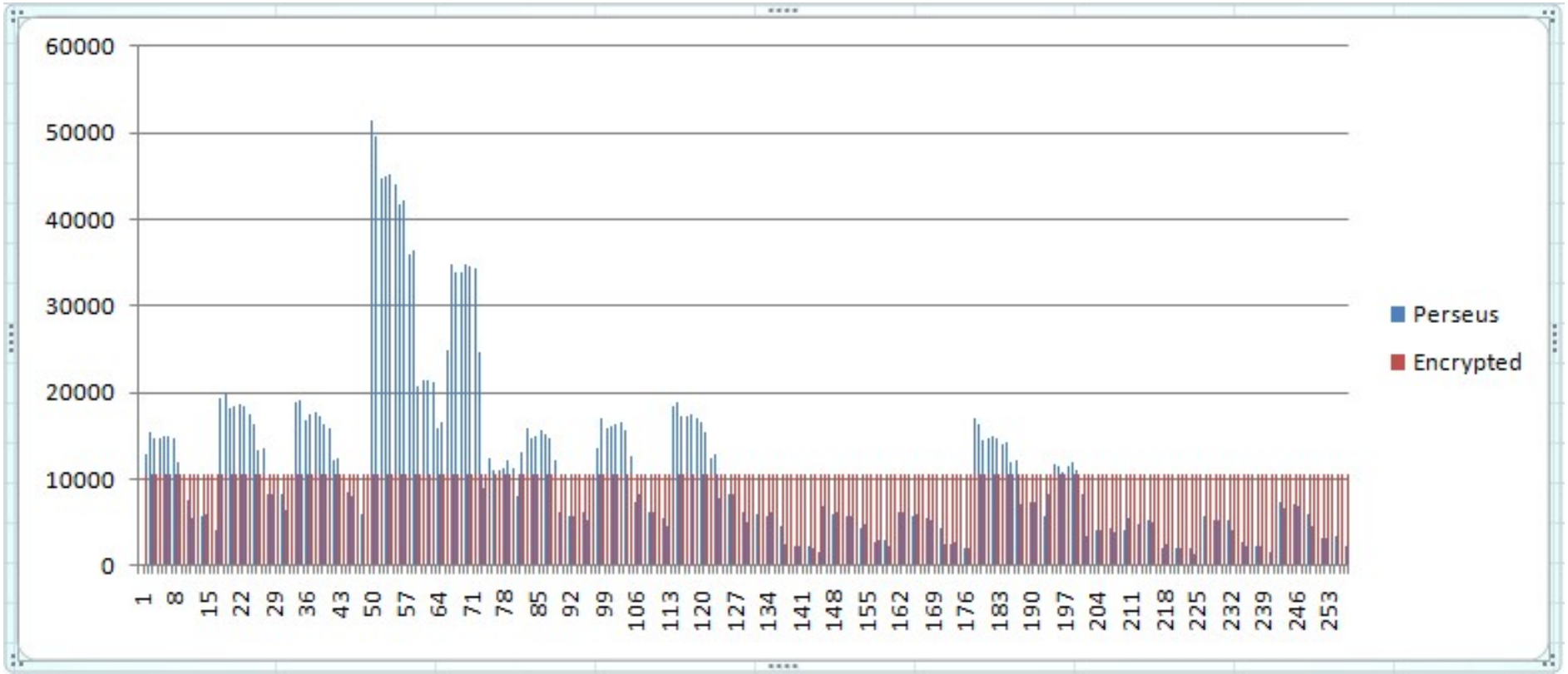
Aspect TRANSEC

Toute donnée protégée par PERSEUS exhibe un profil statistique éloigné de celui de données chiffrées.

Probabilité de bruit	Entropie moyenne texte clair	Entropie moyenne données PERSEUS	Entropie moyenne données chiffrées (AES)
5 %	4.21	4.96	8.00
10 %	4.21	6.19	8.00
15 %	4.21	6.46	8.00
20 %	4.21	7.11	8.00
25 %	4.21	7.39	8.00
30 %	4.21	7.45	8.00
35 %	4.21	7.71	8.00

TABLE: Entropie moyenne pour des textes clair, PERSEUS et AES

Aspect TRANSEC



Aspect TRANSEC

- Le traffic PERSEUS est polymorphe par nature du fait du changement fréquent des paramètres (codeur, bruit...).
- Il n'est donc pas vraiment possible de signer un utilisateur donné.
- Développement du module MIMIC pour simuler n'importe quel traffic ou donnée.
  - Actuellement non public.
  - Aspects légaux non élucidés.

Roadmap 2008 - 2011

- Conception et finalisation théorique du procédé (Eric Filiol)
- Plug-in Firefox pour protection des flux HTTP (Eddy Deligne)
- Librairie Andromède pour protection des flux TORRENT (Fabien Jobin)
- Librairie PERSEUS version 1.x (Eric Filiol)
  - Version 1.6 à venir (nettoyage du code, fix de qq bugs résiduels).
- Port OpenBSD (Pierre-Emmanuel André).
- Application pour protection off-line de fichiers (Jonathan Dechaux et Eric Filiol) : release fin juillet 2011 (phase finale de test et vérification du code).
- Soutien industriel par DFT-Technologies (`http://www.dft-techno.com`).

Roadmap 2011 - 2013

- Version 2.x de la librairie
  - Décodeur en temps/mémoire polynomial.
  - Support de la parallélisation (OpenMP).
- Application VoIP.
- Protection Libre Office version Cloud.
- Application Vidéo et streaming.
- Application Android (SMS, Voix...).
- Carte USB/Ethernet, Ethernet/Ethernet protection mobilité.
- Module MIMIC ? ? ?

Conclusion

- PERSEUS est une technologie élégante qui résout un probème critique :
  - Protéger contre les écoutes abusives, illégales...
  - ... protéger les données privées et le droit fondamental à la vie privée...
  - ... sans crypto...
  - tout en préservant les besoins RÉELS ET LÉGITIMES de sécurité des Etats.
- À ce jour, plus de 200 000 téléchargements (toutes déclinaisons confondues).
- Très nombreux contacts, retours, commentaires... Merci à tous ceux qui nous ont aidé.

Contributeurs et remerciements

- Pierre-Emmanuel André (port OpenBSD), Jonathan Dechaux (application off-line), Eddy Deligne (plug-in Firefox, port Ruby), Guillaume Delugré (analyse implémentation), Anthony Desnos (port Python), Fabien Jobin (Andromède).
- LCL Frédéric Suel (DGSIC), Fondation Libre Office.
- Bowman Wangeci (BAE Systems).
- A tous ceux qui nous ont aidé (conseils, retours, commentaires...) mais qui ont voulu rester anonymes.
- A tous ceux qui utilisent PERSEUS.

Références

- Plug-in Firefox `http://code.google.com/p/perseus-firefox`
- Librairie Andromède `http://code.google.com/p/andromeda`
- Librairie Perseus `http://code.google.com/p/libperseus`
- Eric Filiol (2010). "PERSEUS *Technology : New Trends in Information and Communication Security*". Article Open Access `http://arxiv.org/abs/1101.0057`.

Questions ?